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

Algoritmos e Estruturas de Dados 1 – 2020/2

Lista de Exercícios nº 02 – Recursividade

(Turmas: INF0061/INF0286 – Prof. Wanderley de Souza Alencar)

Sumário

1 Imprimindo números naturais recursivamente **2**

2 Sequência de Fibonacci recursiva **4**

3 Função de Ackermann **8**

4 Reverso de um número natural **10**

5 Conversão de decimal para binário **12**

6 Fatorial duplo **14**

7 O banco inteligente **16**

8 Famílias de Tróia **19**

9 Torre de Hanoi **21**

10 Setas **23**

11 Batalha naval **26**

12 Labirinto – 1 **29**

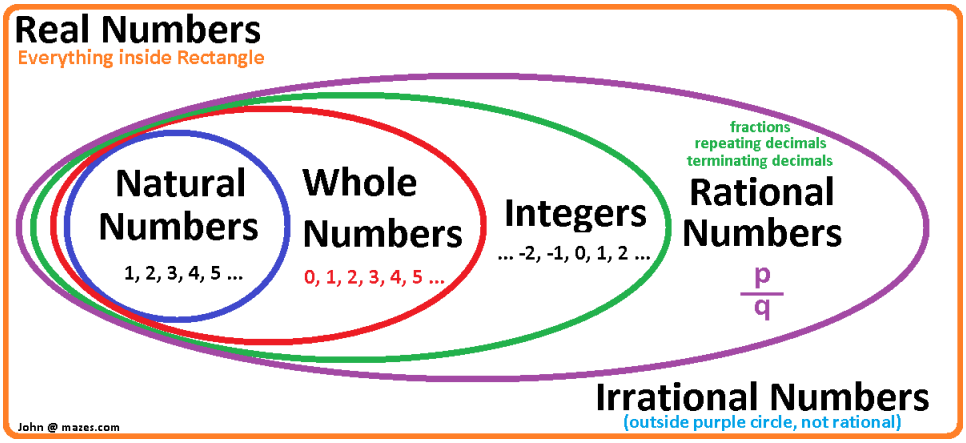
13 Labirinto – 2 **31**

14 Pegar e escapar **33**

15 Altas aventuras **35**

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

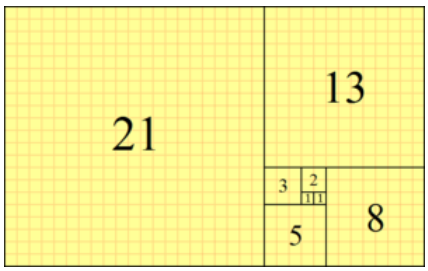
Entrada	Saída
37	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

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 Sequência de Fibonacci recursiva

 (+)



Sem dúvida a chamada "Sequência de Fibonacci" (ou "Sucessão de Fibonacci") é uma das mais famosas sequências numéricas da Matemática. Os dois primeiros termos desta sequência são:
$f_0 = 0$ $f_1 = 1$.
A partir do terceiro termo, cada termo é obtido somando-se os dois termos imediatamente anteriores a ele, ou seja:

$$f_n = f_{n-1} + f_{n-2}, \text{ com } n \in \mathbb{N} \text{ e } n \geq 2$$

Portanto,os seus dez primeiros termos são $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$.

Muitas fontes históricas registram que ela foi "descoberta" (ou "inventada") por Leonardo Fibonacci (1170 – 1250), um matemático italiano que também ficou conhecido como Leonardo de Pisa, sua cidade de nascimento, pois apenas no século XIX ficou associado ao nome *Fibonacci* que, de maneira aproximativa, significa "o filho de Bonacci", em referência ao seu pai Guglielmo dei Bonacci, um próspero mercador.

Aos 32 anos, em 1202, Fibonacci publicou o livro *Liber Abaci* (Livro do Ábaco, ou Livro de Cálculo), um livro de receitas a respeito de como realizar cálculos e que foi o responsável pela disseminação dos números hindu-arábicos na Europa. Num trecho desta obra, Leonardo introduz a sequência por meio de um problema envolvendo *coelhos*. O problema dizia que:

"*Iniciando com um par de coelhos – um macho e uma fêmea – depois de um mês eles de tornam sexualmente adultos e produzem um par de filhotes, também um macho e uma fêmea. Novamente, um mês depois, estes coelhos reproduzem e geram outro par macho-fêmea, os quais, por sua vez, também gerarão outro par macho-fêmea depois de um mês (Claro: ignore aqui a pequeníssima probabilidade de que isto efetivamente ocorra no mundo natural dos* coelhos.)
A questão é: depois de um ano, quantos coelhos haverá?"

A resposta ao problema é obtida por meio do uso da Sequência de Fibonacci – veja "*The 11 most beautiful Mathematical equations*" em:

`https://www.livescience.com/57849-greatest-mathematical-equations.html`.

Uma curiosidade é que, depois disso, Leonardo nunca mais citou a sequência, que ficou esquecida até o século XIX quando matemáticos que trabalhavam em propriedades de sequências numéricas a recuperaram, cabendo ao matemático francês Édouard Lucas (1842 – 1891) ter nomeado, oficialmente, o problema dos coelhos com o nome de "Sequência de Fibonacci".

Entretanto, sabe-se hoje, que Leonardo não descobriu ou inventou a famosa sequência (veja a obra de Keith Devlin intitulada *Finding Fibonacci: The Quest to Rediscover the Forgotten Mathematical Genius*

Who Changed the World, Princeton University Press, 2017), pois citações a ela aparecem em textos antigos, em Sânscrito, muito antes da sua menção por Leonardo.

Vamos a um problema envolvendo-a:

Considere que seja dado um número n, $n \in \mathbb{N}^*$. Usando o conceito de recursividade, elabore um programa em $\mathbb{C}$ para imprimir até o n-ésimo termo da "Série de Fibonacci".

Observação: Note que a contagem dos termos foi iniciada com o termo 0 (zero): $f_0 = 0$.

Entrada

A única linha da entrada contém um número natural n, indicando a ordem máxima dos termos desejados da "Série de Fibonacci". Sabe-se que $1 \leq n \leq 1000$.

Saída

Seu programa deve imprimir uma única linha contendo até o n-ésimo termo da série, sempre separados por um único espaço em branco.

Exemplos

Entrada	Saída
0	0

Entrada	Saída
1	0 1

Entrada	Saída
8	0 1 1 2 3 5 8 13 21

Entrada	Saída
11	0 1 1 2 3 5 8 13 21 34 55 89

Entrada	Saída
15	0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610

Observação: Uma questão interessante é:
Será possível encontrar uma *fórmula* explícita que seja capaz de fornecer um determinado termo da "Sequência de Fibonacci" sem a necessidade de realizar a geração de todos os termos anteriores?
Se isto for possível, gerar utilizar a *fórmula* será mais *eficiente* que utilizar uma função geradora, seja ela recursiva ou iterativa?

Se não for possível, qual o motivo da *impossibilidade*?

3 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

4 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

5 Conversão de decimal para binário

 (++)

Escreva um programa, em $\mathbb{C}$, que receba um número natural $n \in \mathbb{N}$, representado utilizando a notação decimal, e o converta para sua notação binária. O programa deve utilizar uma "*função recursiva*" para realizar a conversão.

Entrada

A primeira linha conterá um número natural estritamente positivo k, $1 \leq k \leq 1000$, que representa o número de casos de teste que virão em seguida.
Cada uma das k linhas seguintes possuem, cada uma, um único número natural, $0 \leq n_i < 10^6$, com $1 \leq i \leq k$, representado utilizando a notação decimal, a ser convertido para sua correspondente representação binária.

Saída

Seu programa deve imprimir k linhas, cada uma com a correspondente representação binária de um número da entrada.

Exemplos

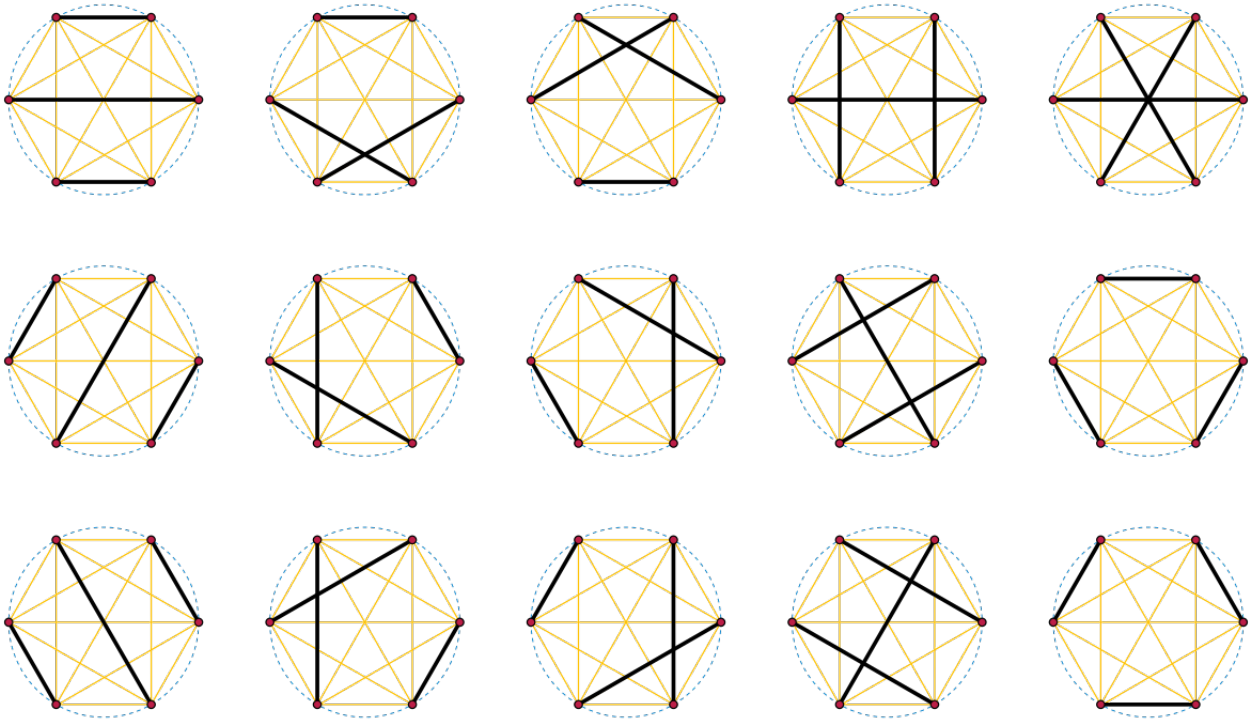
Entrada	Saída
5 1 2 3 4 5	1 10 11 100 101

Entrada	Saída
3 321 753 255	101000001 1011110001 11111111

Entrada	Saída
1 373728	1011011001111100000

6 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

7 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

8 Famílias de Tróia

 (+++)

A *Guerra de Tróia* pode ter sido um grande conflito bélico entre gregos e troianos, possivelmente ocorrido entre os anos de 1.300 a.C. e 1.200 a.C., ou seja, no fim da Idade do Bronze no Mediterrâneo.
Recentemente foram encontradas inscrições numa caverna a respeito de sobreviventes deste conflito e, após um trabalho árduo, arqueólogos descobriram que as incrições descreviam relações de parentesco numa certa população daquela região. Cada item da inscrição indicava duas pessoas que pertenciam a uma mesma família.
O problema dos arqueólogos(as) – que agora é *seu problema* – é determinar quantas famílias distintas existiam naquela população e, obviamente, não eles(as) desejam ter que fazer isto "manualmente", já que computadores existem para estas atividades de agrupamento, dentre outras.

Entrada

O arquivo de entrada consiste de $(m+1)$ linhas.
A primeira linha do arquivo de entrada contém dois números naturais n e m. Onde n, $n \in \mathbb{N}$, indica o número de pessoas na população que, por simplicidade, são sempre numeradas de 1 a n e sabe-se que $1 \leq n \leq 5 \times 10^4$. O valor m indica a quantidade de linhas após a primeira linha, sendo que $m \in \mathbb{N}$ e $1 \leq m \leq 10^5$.
As demais m linhas do arquivo de entrada contêm, cada uma, dois números naturais identificando um par de pessoas daquela população por meio de seus números, sempre separados por um único espaço em branco. Cada linha indica que as duas pessoas pertenciam a uma mesma família.

Saída

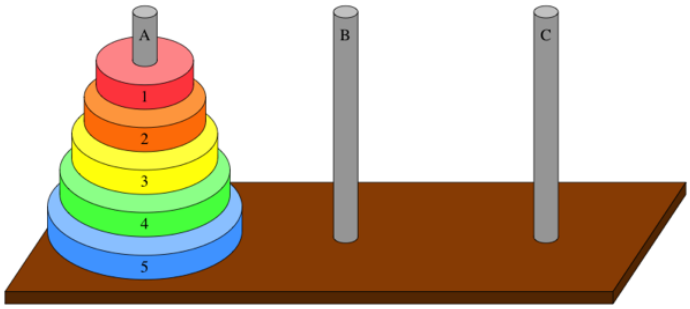
A saída, do programa que será elaborado por você, deve conter apenas uma única linha contendo conte o número de famílias identificadas naquela população.

Exemplos

Entrada	Saída
4 4 1 2 2 3 3 4 4 1	1

Entrada	Saída
8 10 1 2 2 3 3 6 6 5 5 4 4 3 6 7 7 8 8 1 1 5	1

Entrada	Saída
5 4 1 2 2 3 4 5	2



9 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)

10 Setas

 (++++)

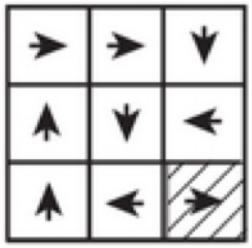
Gabriel é um garoto que gosta muito de um jogo eletrônico onde há várias letras num tabuleiro – que fica sobre o piso – e o jogador precisa, rapidamente, pisar nas letras corretas, de acordo com as instruções que aparecem na *tela de projeção* que está à sua frenta, seguindo uma música ao fundo.

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

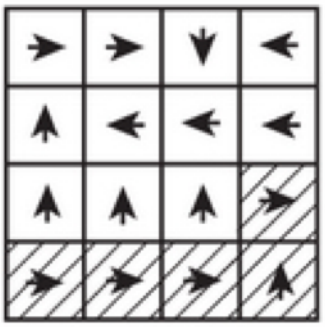
Agora temos um tabuleiro quadrado, com n células de cada lado, em que cada célula possui uma seta que aponta para uma das quatro posições vizinhas (►,◄,▲,▼). O jogador primeiro escolhe uma célula inicial para se posicionar e, quando a música começa, ele deve caminhar na direção para onde a seta em que 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

11 Batalha naval

(++++)

Pedro e Paulo gostam muito de jogar *Batalha Naval*. Apesar de serem grandes amigos, Pedro desconfia que Paulo não esteja jogando *honestamente* e, para tirar essa dúvida, decidiu usar um programa de computador para verificar o resultado de cada jogo.
Acontece que Pedro não sabe programar e, por isso, pediu a sua ajuda para elaborar este *programa de auditoria naval*, explicando-lhe que cada jogador do jogo *Batalha Naval* possui um tabuleiro retangular com n linhas e m colunas ($n, m \in \mathbb{N}^*$) para representar o "campo de batalha", onde:

- cada posição é um quadrado que pode conter `água` ('a') (letra 'a', minúscula) ou uma parte de um `navio` ('#') (um símbolo *hashtag*);
- dois quadrados são ditos *vizinhos* se possuem um lado comum, ou seja, um lado que pertence a ambos quadrados;
- se duas partes de `navio` estão em posições vizinhas, então essas duas partes pertencem ao mesmo navio;
- é proibido que quadrados de duas partes de `navios` distintos tenham um *canto* em comum, ou seja, que quadrados de duas partes de `navios` distintos compartilhem um *vértice*;
- para que um `navio` de um jogador seja destruído por *disparos* de seu oponente é necessário que o oponente acerte todas as partes do `navio`, o que pode exigir um número indeterminado de disparos.

O jogo consiste em *disparos* alternados entre os dois jogadores, sendo que cada disparo que um jogador faz em *direção* ao tabuleiro do seu oponente deve ser feito tendo como *alvo* um único quadrado daquele tabuleiro. Para fazer um *disparo*, um jogador informa ao outro a linha L ($1 \leq L \leq n$) e a coluna C ($1 \leq C \leq m$) do quadrado alvo de seu disparo. Considere que os jogadores não se esquecem de seus disparos anteriores e, por isso, nunca *atiram* no mesmo lugar mais de uma vez.

Sua tarefa é escrever um programa em $\mathbb{C}$ que *simule* uma partida deste jogo a partir da configuração do tabuleiro e de uma sequência de disparos feitos por um dos jogadores, determinando o número de navios do outro jogador que foram destruídos pela sequência de disparos.

Entrada

A primeira linha da entrada contém dois números inteiros n e m ($1 \leq n, m \leq 100$) representando, respectivamente, o número de linhas e de colunas do tabuleiro.
As n linhas seguintes correspondem ao tabuleiro do jogo. Cada uma dessas linhas contém m caracteres, sendo que cada caractere indica o conteúdo da posição correspondente no tabuleiro. Se esse caractere for 'a' (letra 'a', minúscula), essa posição contém `água`; se o caractere for '#' (*hashtag*), essa posição contém uma parte de um `navio`.
A próxima linha contém um número k ($1 \leq k \leq n \times m$) que representa o número de disparos feitos pelo jogador em direção ao tabuleiro de seu oponente.
As próximas k linhas indicam os *disparos* feitos pelo jogador, sendo que cada linha contém dois inteiros L e C, indicando a linha e a coluna do disparo feito, lembrando que $1 \leq L \leq n$ e $1 \leq C \leq m$.

Saída

Seu programa deve imprimir uma única linha contendo um único número natural: o número de navios destruídos do jogador oponente ao que realizou os disparos.

Exemplos

Entrada	Saída
5 5 aa#a# #aaaa aaa#a #aaaa aaa#a 5 1 3 1 4 1 5 2 1 3 4	4

Entrada	Saída
5 5 aa### aaaaa ##### aaaaa #a##a 5 5 1 5 2 1 3 1 4 1 5	2

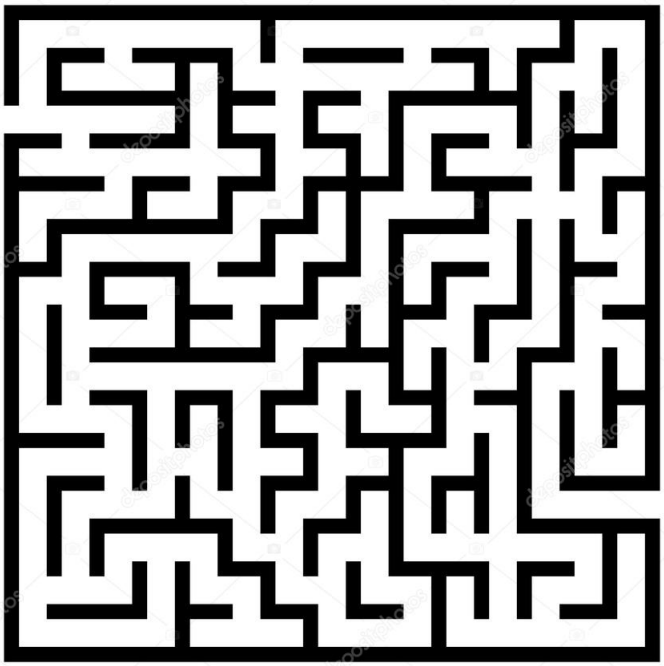
Entrada	Saída
7 7 a#aaaa# ###-a## a#aaaa# aaaa#a# a#aa#a# a####a# aaaaaaa 8 1 1 1 2 2 1 2 2 2 3 3 2 5 2 6 2	1

Observações

Se você domina alguma linguagem de programação que permita o desenvolvimento de programas que apresentem uma GUI (*Graphic User Interface*) para seu usuário, que tal pensar em desenvolver este *joguinho* a utilizando?

12 Labirinto – 1

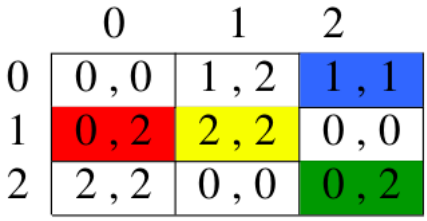
(++++)



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

13 Labirinto – 2

(++++)

Considere que você **continua** jogando o "*Jogo de Labirinto*". Entretanto, desta vez, na versão 2.0, ele ficou um pouco mais complicado, pois a posição inicial não será dada.
Você deve escrever um programa, em $\mathbb{C}$, que dadas as dimensões m e n do labirinto, bem como os respectivos pares de cada casa, como no problema anterior, mas que seja capaz de calcular a "*quantidade de casas*" onde é possível chegar à saída, ou seja, quantas casas permitem que, iniciando-se a partir dela, seja possível ganhar o jogo.
Por exemplo, no tabuleiro a seguir, estão pintadas de vermelho todas as casas que, iniciando-se dela, atinge-se a saída. Neste caso, o programa deveria retornar 4.

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

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.

Saída

Uma única linha com a quantidade de casas a partir da qual é possível alcançar a saída – ganhar o jogo!

Exemplos

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 1 2 1 0 0 2	4

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 1 1 2	9

Entrada	Saída
5 5 0 0 2 0 3 0 1 0 1 1 0 4 3 1 0 2 2 4 2 1 4 4 2 3 1 3 4 2 4 1 0 0 1 4 3 4 2 2 4 3 0 1 4 0 3 2 0 3 1 2	13

14 Pegar e escapar

 (++++)

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

Considere que lhe é fornecido um "vetor" contendo n números naturais, com $1 \leq n \leq 30$.
Você deve escrever um programa, em $\mathbb{C}$, que seja capaz de escolher k números deste vetor de maneira tal que aplicando-se a operação lógica "xor" entre todos os k valores escolhidos, obtenha-se o *máximo* valor possível. Sabe-se que $1 \leq k \leq n$.

A operação "xor" deverá ser aplicada considerando-se a representação binária de cada um dos números.

Entrada

A primeira linha da entrada contém o número de casos de teste, t, a serem submetidos à avaliação. Sabe-se que $1 \leq t \leq 100$.
Cada caso de teste seguinte é formado por $(n+1)$ linhas, onde:

- a primeira linha contém n e k daquele caso de teste, nesta ordem, e separados por um único espaço em branco;
- as n linhas seguintes contém, cada uma, o valor do respectivo elemento do vetor: 1º, 2º, 3º e assim sucessivamente. Sabe-se que o valor de cada elemento está entre 1 e 10000, inclusive extremos.

Saída

Imprima, para cada um dos t casos de teste, uma linha contendo o valor *máximo* obtido para a aplicação da operação "xor" dentre k elementos escolhidos naquele caso de teste.

Exemplo

Entrada	Saída
2 5 3 1 2 3 4 5 5 3 3 4 5 7 4	7 7

Entrada	Saída
2 10 2 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10 2 0 0 0 0 0 1 1 1 1 1	0 1

15 Altas aventuras

 (+++++)

Incentivado por um filme de animação[1], vovô Gepeto resolveu realizar seu sonho de criança: fazer sua pequena casa voar amarrada a balões cheios de gás hélio. Comprou alguns balões coloridos, de boa qualidade, para fazer alguns testes e começou a planejar a sua grande aventura.
A primeira tarefa é determinar qual a quantidade máxima de gás hélio (He) que pode ser injetada em cada balão de maneira tal que ele não *estoure*. Para isto suponha que os valores possíveis de quantidade de gás hélio em cada balão variem, de maneira discreta, entre os valores 1 e n, sendo 1 a mínima quantidade e n a máxima quantidade.
É claro que vovô Gepeto poderia testar "todas as possibilidades" de enchimento dos balões. Evidentemente estetipo de solução ineficiente não é apropriada, ainda mais considerando que vovô Gepeto comprou apenas k balões para os seus testes, com $k \leq n$.
Por exemplo, suponha que $n = 5$ e que $k = 2$. Nesse caso, a melhor solução seria testar o primeiro balão com a quantidade de gás hélio igual a 3. Caso o balão estoure, vovô Gepeto só teria mais um balão, e então teria de testar os valores 1 e 2, no pior caso, somando ao todo três testes. Caso o balão não estoure com o primeiro valor (ou seja, o valor 3 neste caso), vovô poderia testar os valores 4 e depois 5 (ou 5 e depois 4), também somando três testes ao todo.
Observação: Considere que todos os balões tem igual resistência em relação à quantidade de gás hélio que suportam.

Tarefa

Dados a capacidade máxima da bomba disponível para enchimento dos balões ($n \in \mathbb{N}^*$) e o número de balões ($k \in \mathbb{N}^*$), indicar o "número mínimo de testes" que devem ser feitos, no pior caso, para determinar o ponto em que um balão estoura.

Entrada

A única linha da entrada contém dois números naturais, n e k, separados por um único espaço em branco ($1 < k \leq n \leq 1.0 \times 10^6$).

[1] Up! Altas Aventuras, da Pixar Studios, 2009

Saída

Seu programa ℂ deve imprimir uma única linha, contendo um número natural que representa o número mínimo de testes que devem ser feitos, no pior caso, para determinar o ponto em que o balão estoura.

Exemplos

Entrada	Saída
5 2	3

Entrada	Saída
20 2	6

Entrada	Saída
11 5	4