Universidade Federal de Goiás – UFG
Instituto de Informática – INF
Bacharelado em Sistemas de Informação

Algoritmos e Estruturas de Dados 1 – 2018/2

Lista de Exercícios nº 02 – Recursividade
(Turmas: INF0286, INF0061 – Prof. Wanderley de Souza Alencar)

Sumário

1 Imprimir números naturais usando Recursão

 (+)

Escreva um programa, em $\mathbb{C}$, para imprimir os n primeiros números naturais usando recursão.

Entrada

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

Saída

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

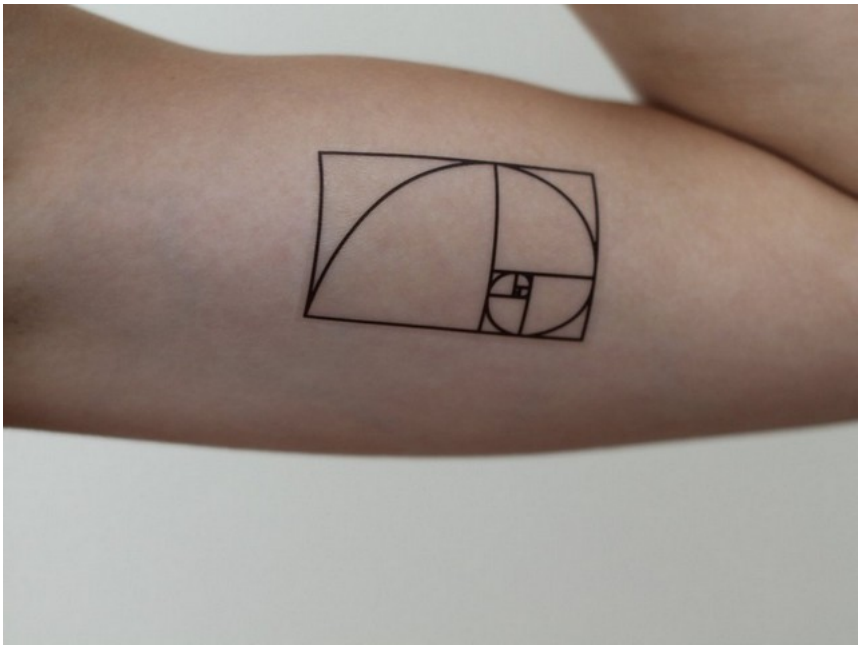
Exemplo

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

2 Encontrar n-ésimo termo da série de Fibonacci usando recursão

 (+)



Dado um número n, $n \in \mathbb{N}^*$, usando recursão, imprimir o n-ésimo termo da série de Fibonacci.]
Observação: Considere que os dois primeiros termos da série são 1 e 1, sendo que a contagem foi iniciada em 1.

Entrada

A única linha da entrada contém um número natural n, indicando a posição do termo desejado na série de Fibonacci.

Saída

Seu programa deve imprimir uma única linha, contendo o n-ésimo termo da série de Fibonacci.

Exemplo

Entrada	Saída
8	21

Entrada	Saída
11	89

Entrada	Saída
15	610

3 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.

Cada jogador de *Batalha Naval* possui um tabuleiro retangular com n linhas e m colunas ($n, m \in \mathbb{N}^*$), onde:

- cada posição é um quadrado que pode conter `água` ('.') ou uma parte de um `navio` ('#');
- dois quadrados são ditos *vizinhos* se possuem um lado comum (ou seja, uma aresta que pertence a ambos quadrados);
- se duas partes de `navio` estão em posições vizinhas, então essas duas partes pertencem ao mesmo navio;
- uma regra do jogo proíbe que os quadrados de duas partes de navios distintos tenham um *canto* em comum (em outras palavras, que quadrados de duas partes de `navios` distintos compartilhem um *vértice*).

Cada disparo que um jogador faz em *direção* ao tabuleiro do seu oponente deve ser feito tendo como *alvo* um único quadrado daquele tabuleiro. 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.
Para que um `navio` seja destruído, o jogador deve acertar todas as partes deste navio. O jogador não pode *atirar* no mesmo lugar mais de uma vez.
Você deve escrever um programa que, dadas a configuração do tabuleiro e uma sequência de disparos feitos por um jogador, determina o número de navios do outro jogador que foram destruídos.

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 '.' (ponto), essa posição contém `água`; se 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. 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 de seu oponente.

Exemplo

Entrada	Saída
5 5 ..#.# #.... ...#. #.... ...#. 5 1 3 1 4 1 5 2 1 3 4	4

Entrada	Saída
5 5 ..### ..... ##### ..... #.##. 5 5 1 5 2 1 3 1 4 1 5	2

Entrada	Saída
7 7 .#....# ###..## .#....# ....#.# .#..#.# .####.# 8 1 1 1 2 2 1 2 2 2 3 3 2 5 2 6 2	1

4 Altas Aventuras

 (+++++)

Incentivado por um filme de animação recente, vovô resolveu realizar seu sonho de criança, fazendo sua pequena casa voar amarrada a balões de hélio. Comprou alguns balões coloridos, de boa qualidade, para fazer alguns testes e começou a planejar a grande aventura. A primeira tarefa é determinar qual a quantidade de hélio máxima que pode ser injetada em cada balão de maneira que ele não estoure. Suponha que os valores possíveis de quantidade de hélio em cada balão variem entre os valores 1 e n. Claro que vovô poderia testar todas as possibilidades, mas esse tipo de solução ineficiente não é apropriada, ainda mais considerando que vovô comprou apenas k balões para os seus testes.
Por exemplo, suponha que $n = 5$ e $k = 2$. Nesse caso, a melhor solução seria testar primeiro o valor 3. Caso o balão estoure, vovô 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 (3, neste caso), vovô poderia testar os valores 4 e depois 5 (ou 5 e depois 4), também somando três testes ao todo.

Tarefa

Dados a capacidade máxima da bomba ($n \in \mathbb{N}^*$) 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 espaço em branco ($1 < k \leq n \leq 1.0 \times 10^9$).

Saída

Seu programa 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.

Exemplo

Entrada	Saída
5 2	3

Entrada	Saída
20 2	6

Entrada	Saída
11 5	4

5 O Banco Inteligente

 (+++)

Caixas automáticos (ATMs – *Automated Teller Machines*) nos bancos são uma ótima invenção mas, às vezes, precisamos de dinheiro *trocado* e a máquina nos entrega notas de $R\$100,00$. Outras 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 $R\$20,00$.
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, dado o valor S do saque (em reais) e quantas notas de cada valor a máquina tem, quantas formas distintas há para entregar o valor S ao cliente.
O `BI` disponibiliza notas de 2, 5, 10, 20, 50 e de 100 reais.
Por exemplo, se $S = 22$ e o número de notas de cada valor é $N_2 = 5$, $N_5 = 4$, $N_{10} = 3$, $N_{20} = 10$, $N_{50} = 0$ e $N_{100} = 10$, então há QUATRO maneiras distintas da máquina entregar o valor do saque:

PRIMEIRA : 20 + 2;

SEGUNDA : 10 + 10 + 2;

TERCEIRA : 10 + 5 + 5 + 2;

QUARTA : 5 + 5 + 5+ 5 + 2.

Tarefa

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, em reais, expressando 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, o número de notas de 2, 5, 10, 20, 50 e 100 reais disponíveis na máquina no momento do saque.

Saída

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

Restrições

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

Exemplo

Entrada	Saída
22 5 4 3 10 0 10	4

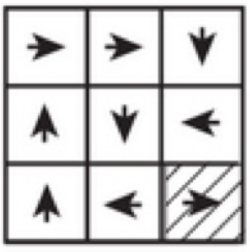
Entrada	Saída
1000 20 20 20 20 20 20	34201

6 Setas

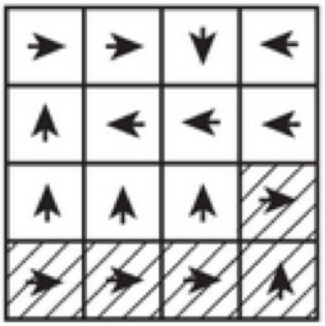
 (++++)

Gabriel é um garoto que gosta muito de um jogo onde há várias letras em um tabuleiro e o jogador precisa rapidamente pisar nas letras corretas, de acordo com as instruções que aparecem na *tela*, seguindo uma música ao fundo. Cansado de vencer, Gabriel inventou um novo jogo: 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 apontar. Ganhará o jogo quem pisar em mais setas corretas durante um determinado período de tempo.

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. Quando isso acontece, dizemos que a célula inicial deste jogo *não é 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 com, respectivamente, oito e onze células *seguras*.



Tabuleiro 3x3 com oito células seguras



Tabuleiro 4x4 com onze células seguras

Ajude Gabriel: dada a configuração do tabuleiro, determine quantas células são *seguras* para ele iniciar o jogo.

Entrada

A primeira linha da entrada contém o número natural n: o tamanho do tabuleiro.
Cada uma das n linhas seguintes contém n caracteres, com as direções das setas. As direções válidas são:

- 'V': 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' 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.

Restrições

- $1 \leq n \leq 500$

Exemplos

Entrada	Saída
3 > > V A V < A < >	8

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

7 Famílias de Tróia

 (+++)

A *Guerra de Tróia* pode ter sido um grande conflito bélico entre gregos e troianos, possivelmente ocorrido entre 1.300 a.C. e 1.200 a.C. (fim da Idade do Bronze no Mediterrâneo). Recentemente foram encontradas inscrições numa caverna a respeito de sobreviventes deste conflito.
Após um trabalho árduo, arqueólogos descobritam que as incrições descreviam relações de parentesco numa certa população. Cada item da inscrição indicava duas pessoas que pertenciam a uma mesma família. O problema – que agora é *seu problema* – é determinar quantas famílias distintas existiam nesta população.

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 indica o número de pessoas na população (numeradas de 1 a n) e m indica a quantidade de linhas após a primeira linha.
As demais m linhas do arquivo de entrada contêm, cada uma, dois números naturais identificando um par de pessoas daquela população.
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.

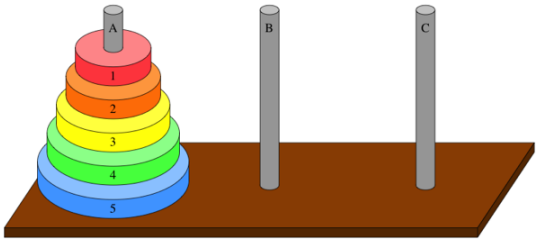
Restrições

- $1 \leq n \leq 5 \times 10^4$
- $1 \leq m \leq 10^5$

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



8 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)

Observação: Note que não há “*espaço em branco*” entre as letras indicativas dos pinos nas respostas, como também para os parênteses.

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

9 Função 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, 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

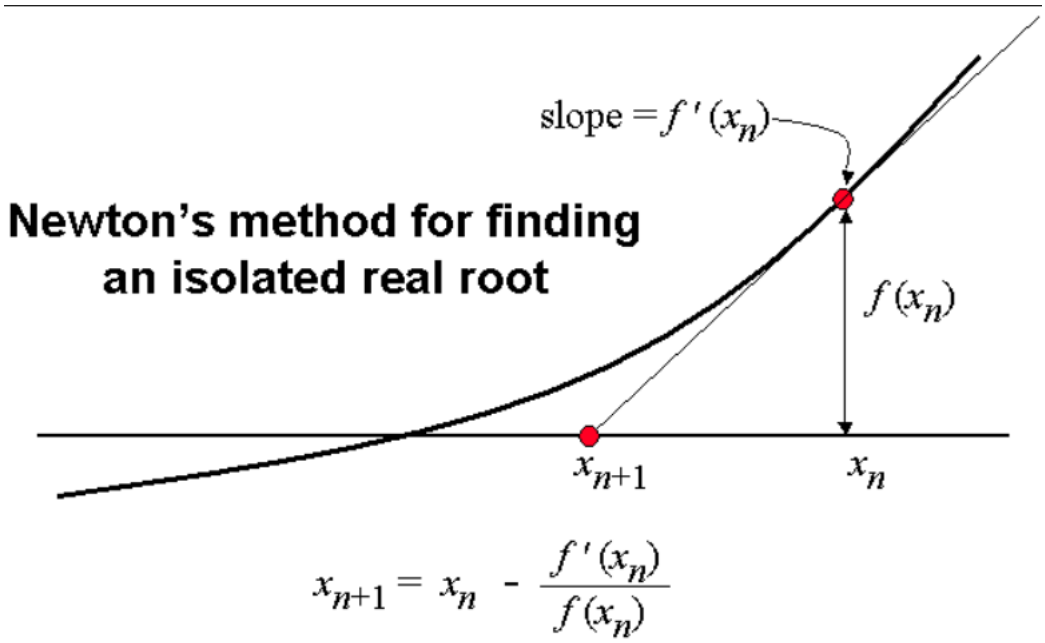
Entrada	Saída
3 0	5

Entrada	Saída
3 2	29

Entrada	Saída
2 4	11

10 Função Raiz Quadrada

 (+)



$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

Um interessante problema matemático é o da elaboração de um algoritmo para determinação da *raiz quadrada* de um número real positivo.
O Método de Newton-Raphson é uma das mais conhecidas soluções. A seguir descrevemos, por meio da definição de uma função recursiva f, a expressão capaz de gerar a raiz quadrada de um número real x positivo passado como parâmetro para a função, sabendo-se que $\sqrt{x} = f(x, x/2)$.

$$f(x,y) = \begin{cases} y, & \text{se } |\, y^2 - x \,| < \varepsilon \\ f\left(x, \frac{\frac{x}{y}+y}{2}\right), & \text{do contrário} \end{cases}$$

Entrada

A única linha da entrada contém dois números reais positivos x e ε, nesta ordem, representando o número para o qual se deseja calcular a raiz quadrada (x) e o valor de tolerância de erro (ε) – tipicamente um número pequeno, como $10^{-6} = 0.000001$.

Saída

Seu programa deve imprimir uma única linha com o valor da raiz quadrada de x encontrada, com SEIS CASAS decimais de precisão.

Exemplos

Entrada	Saída
`2 0.000001`	`1.414214`

Entrada	Saída
3 0.000001	1.732051

Entrada	Saída
28 0.000001	5.291503

11 Reverso de 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

12 Conversão de Decimal para Binário

 (++)

Escreva um programa, em *mathbbC* ou $\mathbb{C}++$, 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

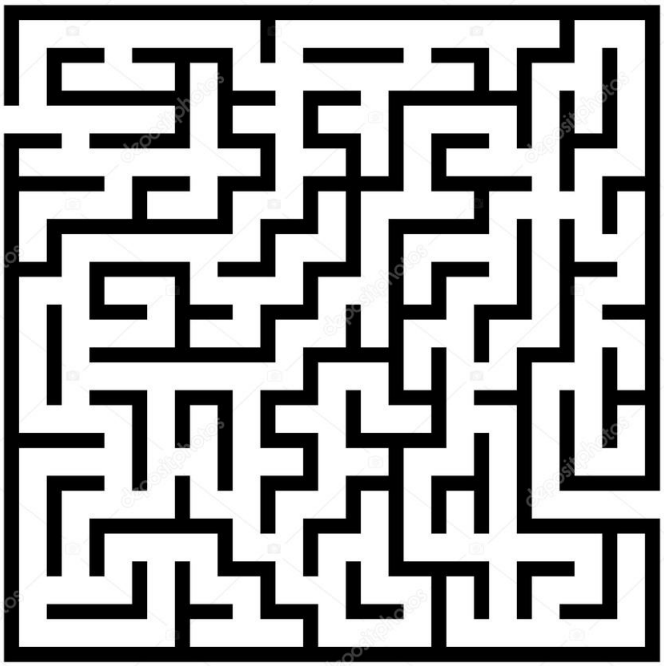
Entrada	Saída
5 1 2 3 4 5	1 10 11 100 101

Entrada	Saída
3 321 753 255	101000001 1011110001 11111111

Entrada	Saída
1 373728	1011011001111100000

13 Labirinto

 (++++)



Considere que você está jogando um "*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.

	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 na saída do labirinto, ou seja, é impossível ganhar o jogo a partir dela.

	0	1	2
0	0 , 0	1 , 2	1 , 1
1	0 , 2	2 , 2	0 , 0
2	2 , 2	0 , 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.
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 SIM (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 NAO (em letras maiúsculas, sem acentuação gráfica), 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	SIM

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 0 0 2 1 0	NAO

14 Labirinto – 2

(++++)

Considere que você **continua** jogando o "*Jogo de Labirinto*". Entretanto, desta vez, ele ficou um pouco mais complicado, pois a posição inicial não será dada.
Você deve escrever um programa, em $\mathbb{C}$, 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 retonar 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	1

15 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 $1 \leq k \leq n$.

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: 1^o, 2^o, 3^o 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