II Maratona Mineira de Programação

Seletiva interna do DCC/UFMG

20 de Abril de 2013

Instruções:

- Este caderno contém 9 problemas. As páginas estão numeradas de 1 a 10, não contando esta página de rosto. Verifique se o caderno está completo.
- Em todos os problemas, a entrada de seu programa deve ser lida da *entrada padrão*. A saída deve ser escrita na *saída padrão*.

Problema A. Anos

Arquivo-fonte: `anos.c`, `anos.cpp` ou `anos.java`

O ano atual, 2013, possui a interessante propriedade de que todos os seus dígitos (2, 0, 1 e 3) são distintos; não há repetição. A última vez em que isso havia ocorrido foi em 1987.

Dados A e B, determine quantos números entre A e B (inclusive) não possuem dígitos repetidos em sua representação decimal.

Entrada

A entrada começa com uma linha contendo apenas um inteiro N, que representa o número de casos de teste ($0 < N \leq 200$). Em seguida, há N linhas, cada uma descrevendo um caso de teste. Cada uma dessas linhas contém dois inteiros A e B ($0 < A \leq B < 3000$).

Saída

Para cada caso de teste, imprima uma linha contendo apenas um inteiro, que representa a quantidade de números no intervalo $[A, B]$ que não possuem dígitos repetidos em sua representação decimal.

Exemplos

anos.in	anos.out
4 1987 2013 2010 2017 1433 2001 6 78	2 5 320 66

Problema B. O Passeio do Rei

Arquivo-fonte: `rei.c`, `rei.cpp` ou `rei.java`

No jogo de Xadrez, há um tabuleiro que é um grid quadricular. O Rei, uma das peças do jogo, pode se mover para qualquer quadrado vizinho, na horizontal, vertical ou diagonal, em um passo.

Suponha que um Rei comece em um quadrado qualquer de um tabuleiro de xadrez que se estende infinitamente em todas as direções. Em quantos quadrados diferentes o Rei pode se encontrar após n passos?

Entrada

Há vários casos de teste. Cada caso de teste é dado em uma linha com um único inteiro n, o número de passos ($0 < n < 16384$). O fim da entrada é indicado por $n = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima o número total de quadrados distintos nos quais o Rei pode se encontrar após n passos.

Exemplos

rei.in	rei.out
4 17 2012 0	81 1225 16200625

Problema C. Caminhos Mínimos

Arquivo-fonte: `caminho.c`, `caminho.cpp` ou `caminho.java`

Uma característica interessante das cidades da Sildávia é que elas, em geral, possuem muitos túneis e viadutos. Devido a isso, várias de suas ruas possuem um limite de altura.

Você foi contratado por uma transportadora. Em cada cidade, há um centro de distribuição. Várias entregas devem ser feitas. Para diminuir os custos, é ideal que o percurso percorrido por cada veículo de entrega seja o menor possível. Porém, obviamente, um veículo não pode passar por uma rua se sua altura for maior que o limite dessa rua. Sua tarefa é determinar qual é o caminho mais curto que um veículo de altura t pode tomar para ir de um ponto a outro da cidade.

As cidades são modeladas como grafos. Cada rua é representada como uma aresta, e os nós representam cruzamentos (ou extremidades, no caso de ruas sem saída). Os nós são numerados, arbitrariamente, de 0 a $n - 1$ (inclusive). O centro de distribuição sempre fica no nó 0. Os destinos das entregas sempre ficam em nós; nunca no meio de uma rua.

Entrada

Há múltiplos casos de teste.

Cada caso de teste começa com uma linha contendo dois inteiros n e m, respectivamente o número de nós e arestas do grafo que representa a cidade ($0 < n \leq 500$, e $0 < m \leq \frac{n \cdot (n-1)}{2}$). Em seguida, há m linhas, cada uma correspondendo a uma rua/aresta. Cada uma delas contém 4 inteiros a, b, d e k. Os inteiros a e b representam os nós que são ligados por essa rua ($0 \leq a, b < n$; $a \neq b$; há no máximo uma rua ligando dois nós). Já d representa o comprimento da rua, em quilômetros ($0 < d \leq 100$) e k representa o limite de altura da rua ($0 \leq k \leq 10$; **se** $k = 0$ **então não há limite de altura para essa rua**).

Em seguida, há uma linha contendo um inteiro c, que representa o número de consultas ($0 < c < 10 \cdot (n - 1)$). Essa linha é seguida por c linhas. Cada uma delas contém dois inteiros w, que é o número do nó onde a entrega deve ser realizada ($0 < w < n$), e t ($0 < t \leq 10$), que é a altura do veículo. Para cada consulta, você deve computar o comprimento do menor caminho que liga o centro de distribuição (no nó 0) ao nó w sem passar por nenhuma rua com limite de altura menor que t, ou determinar que esse caminho não existe.

A entrada termina com $n = m = 0$, que não deve ser processado.

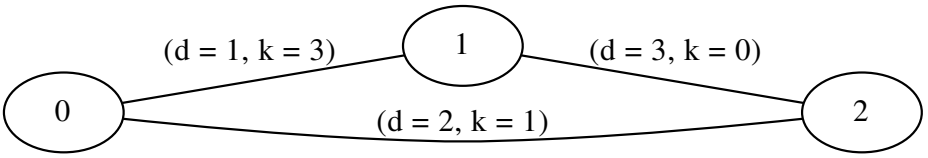
Saída

Para cada caso de teste, imprima uma linha no formato `Teste <x>`, onde `<x>` é o número do caso de teste, começando de 1. Em seguida, para cada consulta no caso de teste imprima uma linha. Essa linha deve conter o comprimento do menor caminho válido para aquela consulta, ou um asterisco (`*`) se não houver nenhum caminho válido. Ao fim de cada caso de teste, inclusive o último, imprima uma linha em branco.

Exemplos

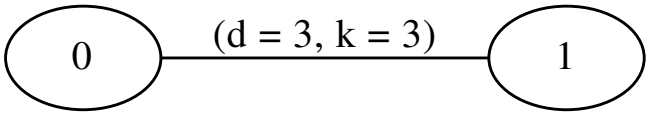
caminho.in	caminho.out
3 3 0 1 1 3 0 2 2 1 1 2 3 0 2 2 1 2 2 2 1 0 1 3 3 2 1 3 1 4 0 0	Teste 1: 2 4 Teste 2: 3 *

O primeiro caso de teste corresponde ao grafo:



As duas consultas se referem ao nó 2. Na primeira delas, a altura do veículo é 1. Isso permite tomar a rua que liga diretamente os nós 0 e 2, e a distância total, portanto, é 2. Na segunda consulta, a altura do veículo é 2. Com isso, fica impossível tomar a rua direta, que possui limite de altura 1. Portanto, é necessário tomar a rua que liga o nó 0 ao 1 (limite 3, distância 1) e em seguida a rua que liga o nó 1 ao 2 (sem limite de altura, distância 3), de forma que a distância total é 4.

Já o segundo caso de teste corresponde ao grafo:



As duas consultas se referem ao nó 1. Na primeira delas, o veículo tem altura 3, e portanto é possível chegar ao nó 1 percorrendo a única rua, que tem distância 3. Na segunda consulta, a altura do veículo é 4. Como o limite de altura da única rua que liga o nó 0 ao 1 é 3, então é impossível que esse veículo chegue ao destino.

Problema D. Árvores Coloridas

Arquivo-fonte: `arvores.c`, `arvores.cpp` ou `arvores.java`

Uma árvore é um grafo não-dirigido acíclico conectado. Sua tarefa nesse problema é colorir árvores, i.e., atribuir uma cor a cada vértice do grafo.

Você pode usar duas cores: azul e branco. Um vértice colorido de azul não pode estar diretamente ligado à outro vértice azul. Vértices brancos podem estar ligados à vértices de qualquer cor.

Uma coloração válida é uma atribuição de cores aos vértices que obedeça as regras acima. Se os n vértices de uma árvore são numerados de 0 a $n-1$, uma coloração pode ser descrita como um vetor de n posições na qual a i-ésima posição indica a cor do vértice i.

Dada uma árvore, calcule o número de colorações válidas distintas para ela. Como esse número pode ser muito grande, calcule-o $\mod 1.000.000.007$.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste começa com uma linha contendo um único inteiro n ($0 < n \leq 262.144$) que indica o número de vértices na árvore. Em seguida, há $n-1$ linhas que contém as arestas da árvore. Cada uma dessas linhas contém dois inteiros a e b, que são os identificadores dos dois vértices ligados por aquela aresta ($0 \leq a, b < n$). Você pode supor que a árvore é válida (i.e., ela é um grafo conectado e acíclico).

A entrada termina com $n = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha na saída contendo o número de colorações válidas distintas para a árvore, $\mod 1.000.000.007$.

Exemplos

arvores.in	arvores.out
1 2 0 1 6 0 1 0 2 0 3 3 4 1 5 0	2 3 22

Uma árvore de apenas um vértice não possui nenhuma aresta, logo não há restrição sobre a cor do vértice. Como há duas possibilidades de cores, há duas colorações possíveis.

Uma árvore de dois vértices necessariamente possui uma aresta os ligando. Logo, os dois não podem ambos serem azuis. Mas qualquer outra combinação (branco+branco, azul+branco, branco+azul) é válida. Portanto, há 3 colorações válidas distintas.

Problema E. Escola Nova

Arquivo-fonte: `escola.c`, `escola.cpp` ou `escola.java`

Tveggja é uma cidade no sul da Sildávia. Nos últimos anos, graças à descoberta de poços de petróleo próximos, a população da cidade vem aumentando em um ritmo alto. Uma das consequências disso é que as escolas da cidade agora estão superlotadas. Para resolver esse problema, a prefeitura quer criar uma nova escola, e precisa da sua ajuda para determinar a localização ideal para ela.

A cidade é um *grid* retangular perfeito. Há m ruas horizontais igualmente espaçadas e n ruas verticais (também igualmente espaçadas). Nos cruzamentos entre duas ruas, há um prédio de vários andares. A prefeitura criou uma lista com os endereços dos alunos que serão transferidos para a nova escola. Como todos os prédios da cidade são localizados nos cruzamentos, é possível especificar todos os endereços por dois números: o número da rua horizontal e o número da rua vertical daquele cruzamento.

A localização ideal para a escola é aquela que minimiza a soma das distâncias que os alunos tem que percorrer para chegar até ela. Lembre-se que há apenas ruas horizontais e verticais. Logo, a distância entre um endereço (x_1, y_1) e um endereço (x_2, y_2) é dada por $|x_1 - x_2| + |y_1 - y_2|$.

Concretamente, se há n alunos e o i-ésimo aluno mora no endereço (x_i, y_i), então a a localização ótima (x_e, y_e) da escola é aquela que minimiza

$$\sum_{i=1}^{n} (|x_i - x_e| + |y_i - y_e|)$$

Observações

- Pode haver mais de um aluno com o mesmo endereço;
- O endereço da escola pode coincidir com o endereço de algum aluno.

Entrada

A entrada é composta por múltiplos casos de teste.

Cada caso de teste começa com uma linha contendo três inteiros m, n e a, respectivamente o número de ruas horizontais, ruas verticais e alunos que serão transferidos para a nova escola ($0 < m, n \leq 1024$, e $0 < a \leq 72319$).

Em seguida, há a linhas. A i-ésima dessas linhas contém dois inteiros x_i e y_i, que representam o endereço do i-ésimo aluno ($0 \leq x_i < m$, e $0 \leq y_i < n$).

A entrada termina com $m = n = a = 0$, que não deve ser processado.

Saída

Para cada caso de teste, determine qual é a menor soma de distâncias dos alunos à escola, e imprima esse valor em uma linha na saída.

Exemplos

escola.in	escola.out
10 10 2 3 3 3 5 0 0 0	2

Problema F. Um Jogo de Cartas

Arquivo-fonte: `baralho.c`, `baralho.cpp` ou `baralho.java`

Pegue todas as 13 cartas de um único naipe do baralho. Embaralhe-as, e as coloque em pilha. Agora, nós vamos jogar um jogo.

Há várias etapas. Em cada etapa, você deve olhar para a carta que está no topo da pilha, e observar qual é o seu valor numérico (ás vale 1, as cartas numeradas valem os seus respectivos números, valete vale 11, dama vale 12 e, por fim, o rei vale 13). Seja n esse valor. Você vai retirar as n primeiras cartas da pilha e as reintroduzir em ordem reversa no topo da pilha. O jogo termina quando a carta do topo da pilha for o às.

Dado o estado inicial da pilha, determine o número de etapas que serão realizadas até que o jogo termine.

Observações

O jogo sempre termina em menos de 4096 etapas, não importa qual seja o estado inicial.

Entrada

A primeira linha da entrada contém um único inteiro t, o número de casos de teste ($0 < t \leq 13123$). Em seguida, há t linhas. Cada linha contém uma permutação das 13 cartas de um naipe do baralho. As cartas numéricas são indicadas por seus respectivos números. O ás é representado pela letra `A` maiúscula, enquanto que valete, dama e rei são representados, respectivamente, pelas letras `J`, `Q` e `K` maiúsculas. Em todos os casos de teste, cada uma das 13 cartas aparece uma e uma única vez. Essa permutação corresponde à descrição da pilha, ordenada do topo ao fundo.

Saída

Para cada caso de teste, imprima uma linha na saída padrão contendo o número de etapas necessárias até que o jogo termine.

Exemplos

baralho.in	baralho.out
3 A 2 3 4 5 6 7 8 9 10 J Q K 6 10 J Q 5 A 9 2 4 K 8 7 3 2 7 A Q 9 6 3 5 J 4 K 10 8	0 1 17

No primeiro caso de teste, o ás já está no topo da pilha, e portanto são necessárias zero etapas até o fim do jogo.

No segundo caso, a pilha inicial contém a carta 6 no topo. Então, retiramos as 6 primeiras cartas, `6 10 J Q 5 A`, e as inserimos na pilha em ordem inversa. Assim, após uma etapa, o estado da pilha será `A 5 Q J 10 6 9 2 4 K 8 7 3`, que contém o ás no topo. Portanto, o jogo termina após uma etapa.

Problema G. Campus da UFMG

Arquivo-fonte: `campus.c`, `campus.cpp` ou `campus.java`

Com o fim da Copa do Mundo de 2014, onde o Uruguai sagrou-se campeão, o campus da UFMG agora é um ótimo lugar para as pessoas passearem aos domingos, graças às obras de revitalização que o tornaram o espaço público mais belo da cidade. O novo campus foi modelado como uma matriz de dimensões N por M, onde cada uma das NxM células pode ser calçamento, por onde as pessoas podem andar, ou área verde, onde é proibido pisar. O calçamento foi projetado de forma que todos os blocos de concreto estejam conectados, ou seja, partindo de qualquer bloco é possível alcançar todos os outros blocos sem precisar pisar na grama. Dois blocos são considerados conectados se são adjacentes vertical ou horizontalmente.

Agora que o país está falido e não existem mais investimentos, a reitoria da universidade está preocupada com a manutenção desse espaço. Como os blocos de concreto são caros e a maior parte do orçamento anual já está reservada para os times que disputarão a Maratona de Programação este ano, o reitor deseja saber qual o menor número de blocos que precisam ser danificados para que o calçamento deixe de ser conexo.

Observações

Calçamentos constituído por zero ou um bloco são considerados conectados por definição.

Entrada

A entrada possui vários casos de teste. Cada caso começa com dois inteiros N e M ($1 \leq N, M \leq 300$), indicando respectivamente o número de linhas e de colunas da matriz que representa o campus. Cada uma das N linhas que seguem contém M caracteres cada, onde # representa um bloco de calçamento e . representa um bloco de área verde. A saída termina com o final do arquivo.

Saída

Para cada caso de teste, imprima uma linha contendo um único inteiro, a quantidade mínima de blocos que precisam ser danificados para desconectar ao menos um bloco de calçamento dos demais. Caso não seja possível obter mais de um grupo conexo de blocos, imprima -1.

Exemplos

campus.in	campus.out
3 3 .#. .#. .#. 3 3 ### #.# ###	1 2

Problema H. Panquecas da Tia Ju

Arquivo-fonte: `panquecas.c`, `panquecas.cpp` ou `panquecas.java`

As panquecas da Tia Ju sempre fizeram sucesso entre a família. Após hesitar por muito tempo, ela decidiu abrir uma panquecaria. Que foi um sucesso enorme. Tão enorme que agora ela está com grande dificuldade em atender a todos os pedidos em tempo hábil.

Uma panqueca precisa ser frita dos dois lados, por 1 minuto de cada lado. A única frigideira de Tia Ju comporta duas panquecas simultaneamente (e apenas duas). Ela recebeu vários pedidos de panquecas. E vai processar cada pedido em ordem, de forma independente (isso é, ela só começará a preparar as panquecas de um pedido depois que as panquecas de todos os pedidos anteriores já estiverem prontas, mesmo que isso implique que em algum momento a frigideira fique com apenas uma panqueca.

São dados o número de pedidos, e o número de panquecas requisitadas em cada pedido. Determine qual é o menor tempo necessário para atender à todos os pedidos seguindo as regras descritas acima.

Observações

- Os pedidos devem ser processados em ordem.

Entrada

Há vários casos de teste. Cada caso começa com uma linha contendo um único inteiro k, o número de pedidos ($0 < k \leq 32768$). Em seguida, há uma linha contendo k inteiros $p_1, p_2, \ldots, p_k$, onde p_i representa o número de panquecas requisitadas no i-ésimo pedido ($0 < p_i \leq 1024$, para todo i).

A entrada termina com $k = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha na saída contendo um único inteiro, que representa o menor tempo necessário para atender à todos os pedidos.

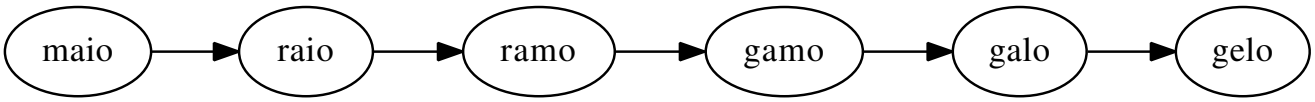
Exemplos

panquecas.in	panquecas.out
2 2 2 0	4

Problema I. Cadeias de Palavras

Arquivo-fonte: `cadeia.c`, `cadeia.cpp` ou `cadeia.java`

Uma cadeia de palavras é uma sequência de palavras tal que todas elas possuem o mesmo comprimento e duas palavras consecutivas diferem apenas por uma letra. Por exemplo:



Dada uma palavra, determine quais palavras (de um dicionário dado) podem sucedê-la numa cadeia.

Entrada

A entrada começa com uma linha contendo apenas um inteiro N (considere que $0 < N \leq 10^4$), que representa o número de palavras no dicionário. Em seguida, há N linhas, cada uma contendo uma palavra. Cada palavra é composta apenas por letras maiúsculas, sem acentuação ou espaços. Não há palavra com mais de 10 caracteres. Não há palavras repetidas.

Após o dicionário, há uma linha contendo apenas um inteiro M ($0 < M \leq 10^4$). Após essa linha, há M linhas, cada uma das quais contém uma palavra do dicionário. Para cada uma dessas palavras, você deve determinar a lista de palavras do dicionário que podem sucedê-la numa cadeia.

Saída

Para cada uma das M palavras, imprima uma linha na saída no formato `<palavra>: <sucessor_1> <sucessor_2> ... <sucessor_k>`, onde `<palavra>` é a palavra original, e cada `<sucessor_i>` é uma palavra do dicionário que possui o mesmo tamanho da palavra original e difere desta por apenas uma letra. Os sucessores devem estar ordenados em ordem lexicográfica (ordem do dicionário).

Exemplos

cadeia.in	cadeia.out
10 MUCO LIXO MACA FOGO FOGE MAIO RAIO MAGO RAMO MAGRO 3 MAIO FOGO LIXO	MAIO: MAGO RAIO FOGO: FOGE LIXO:

De 'MAIO', é possível chegar em 'MAGO' trocando o I por G, e em 'RAIO' trocando o M por R. De 'FOGO', chega-se em 'FOGE' trocando o segundo O por E. Nenhuma das 10 palavras no dicionário pode ser formada trocando apenas uma das letras de 'LIXO'.