Competidor(a): _____

Número de inscrição: _____–_____ *(opcional)*

OBI2019

Caderno de Tarefas

Modalidade **Programação** • **Nível 1** • Fase **Nacional**

21 de setembro de 2019

A PROVA TEM DURAÇÃO DE **4** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 8 páginas (não contando a folha de rosto), numeradas de 1 a 8. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa; em particular, seu programa não deve escrever frases como "Digite o dado de entrada:" ou similares.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão necessariamente ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as tarefas mais fáceis.
- Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. Soluções na linguagem C devem ser arquivos com sufixo *.c*; soluções na linguagem C++ devem ser arquivos com sufixo *.cc* ou *.cpp*; soluções na linguagem Pascal devem ser arquivos com sufixo *.pas*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; soluções na linguagem Python 2 devem ser arquivos com sufixo *_py2.py*; soluções na linguagem Python 3 devem ser arquivos com sufixo *_py3.py*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*.
- Na linguagem Java, **não** use o comando *package*, e note que o nome de sua classe principal deve usar somente letras minúsculas (o mesmo nome do arquivo indicado nas tarefas).
- Para tarefas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada tarefa.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou pen-drive, conforme especificado pelo seu professor.
- Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
  - em Pascal: *readln, read, writeln, write*;
  - em C: *scanf, getchar, printf, putchar*;
  - em C++: as mesmas de C ou os objetos *cout* e *cin*.
  - em Java: qualquer classe ou função padrão, como por exemplo *Scanner, BufferedReader, BufferedWriter* e *System.out.println*
  - em Python: *read,readline,readlines,input,print,write*
  - em Javascript: *scanf, printf*
- Procure resolver a tarefa de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

Coleção de Upas

Nome do arquivo: `colecao.c`, `colecao.cpp`, `colecao.pas`, `colecao.java`, `colecao.js`, `colecao_py2.py` *ou* `colecao_py3.py`

Mayuri é uma jovem que adora colecionar Upas. Ela está sempre procurando pelos melhores Upas para melhorar sua coleção. Cada Upa possui uma cor única e como Mayuri é muito perfeccionista ela não acha que todas cores combinam juntas, então ela resolveu escrever uma lista com pares de cores que não combinam. No entanto, ela está muito confusa em como organizar sua coleção, pois existem Upas mais raros que outros e por isso ela também precisa manter sempre os Upas mais raros.

Sua coleção é composta por N Upas e ela possui exatamente um Upa de cada cor entre 1 e N. Um Upa de cor i possui raridade igual a 2^i. Dada a coleção atual de Upas de Mayuri, informe quais Upas ela deve manter na sua coleção de modo que todos os Upas possuem cores que combinam entre si e tal que a soma das raridades de todos os Upas é maior possível.

Entrada

A primeira linha da entrada contém dois números inteiros N e M, indicando respectivamente o número de Upas e o tamanho da lista de pares de cores que não combinam. As próximas M linhas contêm, cada uma, dois inteiros U e V, indicando que as cores U e V não combinam.

Saída

Seu programa deve produzir duas linhas de saída. A primeira linha da saída é composta por um inteiro Q indicando a quantidade de Upas que Mayuri deve manter na coleção. A segunda linha da saída deve ser composta por Q inteiros, indicando quais Upas ela manter na coleção, **em ordem crescente de cor**.

Restrições

- $1 \leq N, M \leq 10^5$.
- $1 \leq U, V \leq N$ e $U \neq V$
- Mayuri possui exatamente um Upa para cada cor entre 1 e N
- É garantido que existe exatamente uma única resposta

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 20 pontos, $1 \leq N \leq 10$ e $1 \leq M \leq 15$.
- Para um conjunto de casos de testes valendo outros 20 pontos, $1 \leq N \leq 15$ e $1 \leq M \leq 30$.
- Para um conjunto de casos de testes valendo outros 20 pontos, $1 \leq N, M \leq 1000$.
- Para um conjunto de casos de testes valendo outros 40 pontos, não existem restrições adicionais.

Exemplo de entrada 1	**Exemplo de saída 1**
10 4 10 9 8 7 8 6 1 2	6 2 3 4 5 8 10

Exemplo de entrada 2	**Exemplo de saída 2**
13 19 12 1 12 2 12 3 12 4 10 5 13 6 3 7 1 8 1 9 11 10 7 11 12 13 1 5 9 13 6 2 8 11 8 7 11 3 7 12	5 2 4 5 11 13

Linhas de Ônibus

Nome do arquivo: `linhas.c`, `linhas.cpp`, `linhas.pas`, `linhas.java`, `linhas.js`, `linhas_py2.py` *ou* `linhas_py3.py`

Nessa grande cidade na China, há T terminais de ônibus, numerados de 1 a T; e L linhas de ônibus,numeradas de 1 a L. Os mapas são muito confusos mas conseguimos entender que os ônibus de uma linha fazem viagens circulares passando por um conjunto fixo de terminais. Por exemplo, a tabela seguinte indica o conjunto de terminais por onde passam os ônibus de cada linha, para $T = 10$ e $L = 5$:

Linha	Conjunto de Terminais
1	$\{4, 3, 8, 2, 1\}$
2	$\{5, 10, 7\}$
3	$\{1, 5\}$
4	$\{6, 8, 10\}$
5	$\{9, 4, 5\}$

Não estamos preocupados com o trajeto da linha, com a ordem na qual o ônibus passa pelos terminais. Portanto, para ir do terminal 2 para o terminal 4, precisamos apenas tomar um ônibus da linha 1 e esperar até ele chegar no terminal 4. O sistema garante que é possível viajar entre qualquer par de terminais, mas talvez seja preciso trocar de linha de ônibus algumas vezes.

Nós estamos com medo de tomar um ônibus errado e acabar perdidos na cidade. É tudo muito grande na China! Por isso, queremos trocar de ônibus o menor número possível de vezes. Por exemplo, você pode ir do terminal 2 para o terminal 10 primeiro tomando a linha 1 até o terminal 1, depois a linha 3 até o terminal 5 e, por fim, a linha 2 até o terminal 10; trocando de ônibus duas vezes, usando três linhas no total. Só que dá para ir do terminal 2 para o 10 trocando apenas uma vez: primeiro tomando a linha 1 até o terminal 8 e depois a linha 4 até o terminal 10.

Neste problema, dados os conjuntos de terminais de cada linha, um terminal origem e um terminal destino, seu programa deve computar o número mínimo possível de linhas de ônibus para fazer a viagem.

Entrada

A primeira linha da entrada contém quatro inteiros, T, L, O e D, representando, respectivamente, o número de terminais, o número de linhas de ônibus, o terminal origem e o terminal destino. As últimas L linhas da entrada descrevem, cada uma, o conjunto de terminais pelos quais uma linha de ônibus passa. A i-ésima linha (dessas últimas L linhas da entrada) descreve o conjunto de terminais da linha de ônibus i, no seguinte formato: o primeiro inteiro na linha, C, indica o número de terminais no conjunto. Depois desse inteiro, o restante da linha da entrada contém C inteiros distintos representando os terminais.

Saída

Seu programa deve produzir uma única linha, contendo apenas um inteiro, o número mínimo possível de linhas de ônibus para viajar do terminal O para o terminal D.

Restrições

- $2 \leq T \leq 500$
- $1 \leq L \leq 500$
- $2 \leq C \leq T$
- $O \neq D$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 5 pontos, $L = 2$
- Em um conjunto de casos de teste somando outros 5 pontos, $T = 3$
- Em um conjunto de casos de teste somando outros 10 pontos, $T \leq 10$
- Em um conjunto de casos de teste somando outros 20 pontos, $T \leq 100$
- Em um conjunto de casos de teste somando outros 20 pontos, $C \leq 10$
- Em um conjunto de casos de teste somando os demais 40 pontos, nenhuma restrição adicional

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
10 5 2 10 5 4 3 8 2 1 3 5 10 7 2 1 5 3 6 8 10 3 9 4 5	2

Exemplo de entrada 2	**Exemplo de saída 2**
2 1 1 2 2 2 1	1

Exemplo de entrada 3	**Exemplo de saída 3**
10 9 1 10 2 1 2 2 2 3 2 3 4 2 4 5 3 5 6 7 2 6 7 2 7 8 2 8 9 2 9 10	8

Parcelamento sem juros

Nome do arquivo: `parcelamento.c`, `parcelamento.cpp`, `parcelamento.pas`, `parcelamento.java`, `parcelamento.js`, `parcelamento_py2.py` *ou* `parcelamento_py3.py`

Pedrinho está implementando o sistema de controle de pagamentos parcelados de uma grande empresa de cartão de crédito digital. Os clientes podem parcelar as compras sem juros no cartão, em até 18 vezes. Quando o valor V da compra é divisível pelo número P de parcelas que o cliente escolhe, todas as parcelas terão o mesmo valor. Por exemplo, se o cliente comprar um livro de $V = 30$ reais em $P = 6$ vezes, então as parcelas terão valores: 5, 5, 5, 5, 5 e 5. Mas se o valor da compra não for divisível pelo número de parcelas será preciso fazer um ajuste, pois a empresa quer que todas as parcelas tenham sempre um valor inteiro e somem no total, claro, o valor exato da compra. O que Pedrinho decidiu foi distribuir o resto da divisão de V por P igualmente entre as parcelas iniciais. Por exemplo, se a compra for de $V = 45$ e o número de parcelas for $P = 7$, então as parcelas terão valores: 7, 7, 7, 6, 6, 6 e 6. Quer dizer, como o resto da divisão de 45 por 7 é 3, então as 3 parcelas iniciais devem ter valor um real maior do que as 4 parcelas finais.

Você precisa ajudar Pedrinho e escrever um programa que, dado o valor da compra e o número de parcelas, imprima os valores de cada parcela.

Entrada

A primeira linha da entrada contém um inteiro V, representando o valor da compra. A segunda linha da entrada contém um inteiro P, indicando o número de parcelas.

Saída

Seu programa deve imprimir P linhas, cada uma contendo um inteiro representando o valor de uma parcela. A i-ésima linha deve conter o valor da i-ésima parcela, para $1 \leq i \leq P$, de acordo com o que Pedrinho decidiu.

Restrições

- $10 \leq V \leq 1000$
- $2 \leq P \leq 18$

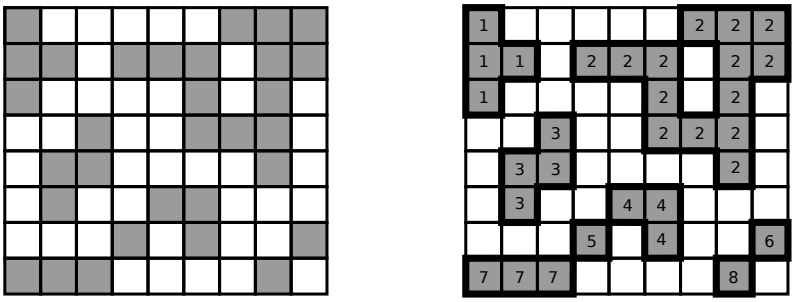
Exemplo de entrada 1	**Exemplo de saída 1**
30 6	5 5 5 5 5 5

Exemplo de entrada 2	**Exemplo de saída 2**
45 7	7 7 7 6 6 6 6

Manchas de pele

Nome do arquivo: `manchas.c`, `manchas.cpp`, `manchas.pas`, `manchas.java`, `manchas.js`, `manchas_py2.py` *ou* `manchas_py3.py`

O laboratório de dermatologia da Linearlândia está implementando um software para contar o número de manchas presentes numa imagem digital de N por M pixels. Cada pixel na imagem é preto ou branco e dois pixels pretos distintos A e B pertencem à mesma mancha se e somente se: existir uma sequência de pixels $[P_1, P_2, \ldots, P_k]$, onde $k \geq 2$, $A = P_1$, $B = P_k$ e para todo $1 \leq i < k$, P_i é ortogonalmente adjacente a P_{i+1} (P_i imediatamente acima, abaixo, à esquerda ou à direita de P_{i+1}).



A figura acima, para $N = 8$ e $M = 9$, ilustra uma imagem digital onde existem oito manchas. Dada a imagem, seu programa deve contar o número de manchas presentes.

Entrada

A primeira linha da entrada contém dois inteiros N e M, representando, respectivamente, o número de linhas e colunas da imagem. As N linhas seguintes contêm, cada uma, M inteiros P representando os pixels da imagem.

Saída

Seu programa deve imprimir uma linha contendo um inteiro, o número de manchas na imagem.

Restrições

- $1 \leq N \leq 1000$
- $1 \leq M \leq 1000$
- O valor de P é 1, representando um pixel preto, ou 0, representando um pixel branco.

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 10 pontos, $N = M = 2$.
- Para um conjunto de casos de testes valendo outros 20 pontos, $N = 1$.
- Para um conjunto de casos de testes valendo outros 20 pontos, $N, M \leq 100$.
- Para um conjunto de casos de testes valendo outros 50 pontos, nenhuma restrição adicional (*Atenção, para essa parcial, não é recomendada uma implementação recursiva!*)

Exemplo de entrada 1	**Exemplo de saída 1**
8 9 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0	8

Exemplo de entrada 2	**Exemplo de saída 2**
1 1 0	0

Exemplo de entrada 3	**Exemplo de saída 3**
1 10 0 0 1 0 1 1 1 0 1 0	3