OBI2014

Caderno de Tarefas

Modalidade **Universitária**, Fase **2**

16 de agosto de 2014

A PROVA TEM DURAÇÃO DE **5** HORAS

Promoção:

Sociedade Brasileira de Computação

Patrocínio:

Fundação Carlos Chagas

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 10 páginas (não contando a folha de rosto), numeradas de 1 a 10. 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.
- 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 ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões 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; e soluções na linguagem Python devem ser arquivos com sufixo *.py*. Para problemas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada problema.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou disquete, 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,print,write*
- Procure resolver o problema 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.

Notas

Nome do arquivo fonte: `notas.c`, `notas.cpp`, `notas.pas`, `notas.java`, *ou* `notas.py`

O professor Arquimedes precisa da sua ajuda para descobrir qual é a nota mais frequente entre as notas que os alunos dele tiraram na última prova. A turma tem N alunos e seu programa deve imprimir a nota que aparece mais vezes na lista de N notas. Se houver mais de uma nota mais frequente, você deve imprimir a maior delas! Por exemplo, se a turma tiver $N = 10$ alunos e as notas forem $[20, 25, 85, 40, 25, 90, 25, 40, 55, 40]$, as notas mais frequentes são 25 e 40, ocorrendo três vezes cada. Seu programa, então, deve imprimir 40.

Entrada

A entrada consiste de duas linhas. A primeira linha contém um número inteiro N, o número de alunos na turma. A segunda linha contém N inteiros, que é a lista de notas dos alunos.

Saída

Seu programa deve imprimir apenas uma linha contendo apenas um número, a nota mais frequente da lista.

Restrições

- $1 \leq N \leq 200$
- O valor de todas as notas é um inteiro entre 0 e 100, inclusive

Exemplos

Entrada	**Saída**
10 20 25 85 40 25 90 25 40 55 40	40

Entrada	**Saída**
12 45 0 33 70 12 55 70 70 90 55 70 100	70

Voo

Nome do arquivo fonte: `voo.c`, `voo.cpp`, `voo.pas`, `voo.java`, *ou* `voo.py`

João estava navegando na internet, olhando horários de voos de várias companhias aéreas entre diferentes cidades, e notou que em alguns casos voos diretos entre duas cidades tinham tempos diferentes, dependendo se eram voos de ida ou de volta.

A única explicação possível era a de voos ligando cidades localizadas em diferentes fusos horários. João então chegou à conclusão que seria possível determinar a diferença entre os fusos horários, com base apenas nos horários fornecidos pelas companhias aéreas.

Por exemplo, um voo sai da Haquérnia às 10:00 horas e chega na Nerdínia às 22:00 horas, ao passo que outro voo sai da Nerdínia às 10:00 horas e chega na Haquérnia às 18:00 horas. Qual a explicação? Note que ambos os voos utilizam aeronaves idênticas, na mesma rota, um de ida, outro de volta. Na realidade, o voo dura 10 horas e Nerdínia fica em um fuso horário +2 horas à frente do fuso horário da Haquérnia (portanto o fuso horário de Haquérnia fica −2 horas à frente do fuso horário de Nerdínia).

João anotou então a tabela de horários de várias companhias aéreas, porém cometeu um engano. Esqueceu-se de anotar datas de partida e chegada. Por exemplo, se a partida de um voo é às 18:00 e a chegada é às 14:00, João não sabe dizer se a data de chegada é a seguinte à da partida, em voo que dura 20 horas, entre cidades no mesmo fuso horário, ou se a data de chegada é a mesma da de partida, com uma duração de voo de uma hora, em que a cidade destino está em fuso horário com cinco horas a menos do fuso horário da cidade origem.

Sua tarefa é ajudar João a determinar a duração do voo e a diferença entre os fusos horários de chegada e de partida de cada par de voos da tabela, um de ida outro de volta, mesmo sem saber as datas dos voos.

Entrada

A entrada é composta de apenas uma linha, com 4 horários, separados por um espaço em branco. Esses horários envolvem voos entre duas cidades, A e B e são, respectivamente, p_A, c_B, p_B e c_A. O horário p_A indica a hora da partida de um voo de A para B, hora local de A. O horário c_B indica a hora de chegada do mesmo voo na cidade B, hora local de B. O horário p_B é a hora de partida do voo de volta, de B para A, hora local de B. O horário c_A é a hora de chegada do voo de volta, hora local de A.

Saída

A saída consiste de uma linha, informando a duração do voo em minutos e quantas horas B está à frente de A, em termos de fusos horários. Os dois valores devem ser separados por um espaço em branco.

Restrições

- Todos os horários são da forma `h:m`, em que $0 \leq h < 24$ e $0 \leq m < 60$.
- A duração de cada voo é inferior a 12 horas.
- A diferença δ entre dois fusos horários é sempre um número inteiro de horas, no intervalo $-12 < \delta \leq 12$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 30 pontos, as datas de partida e de chegada de cada voo são iguais.

Exemplos

Entrada	**Saída**
10:00 22:00 10:00 18:00	600 2

Entrada	**Saída**
17:00 23:00 17:00 13:00	60 5

Entrada	**Saída**
10:00 18:00 10:00 22:00	600 -2

Entrada	**Saída**
17:00 13:00 17:00 23:00	60 -5

Entrada	**Saída**
18:00 12:00 18:00 14:00	420 11

Entrada	**Saída**
18:00 14:00 18:00 12:00	420 -11

Blefe

Nome do arquivo fonte: `blefe.c`, `blefe.cpp`, `blefe.pas`, `blefe.java`, *ou* `blefe.py`

Pedro está desenvolvendo um jogo on-line para dois jogadores, em que o objetivo é forçar um erro do adversário, blefando. A questão é que, à medida que o jogo prossegue, mais tempo é necessário para verificar se uma jogada é válida ou não, ou seja, se é um blefe ou não. Daí que Pedro precisa da sua ajuda para implementar um algoritmo rápido para verificar se uma jogada é ou não um blefe.

Considere um conjunto A fixo de N números inteiros, positivos ou negativos, e uma sequência de números inteiros B, inicialmente vazia. Os jogadores se alternam em jogadas que consistem em incluir um número por vez no final da sequência B. Quando chega a sua vez, um jogador deve fazer uma de duas jogadas válidas possíveis: *(i)* incluir em B qualquer um dos números do conjunto A; *(ii)* ou incluir em B um número que é a soma de dois números quaisquer que já estejam em B (note a soma não é de números necessariamente distintos, pode ser a soma de um número com ele mesmo).

Nesta tarefa, você deve escrever um programa que, dado o conjunto A e uma sequência B, diga se todas as jogadas foram válidas, ou mostre qual é a primeira jogada inválida em B.

Entrada

A entrada consiste de três linhas. A primeira linha contém dois números N e M, respectivamente o tamanho do conjunto A e o tamanho da sequência B. A segunda linha contém os N números inteiros do conjunto A. A terceira linha contém os M números inteiros da sequência B.

Saída

Seu programa deve produzir uma única linha. A linha deve conter a palavra "`sim`" caso todas as jogadas em B sejam válidas; se houver alguma jogada inválida em B, a linha deve conter o primeiro número inválido em B.

Restrições

- $1 \leq N \leq 10^3$ e $1 \leq M \leq 10^4$
- O valor de todos os números em A e em B está entre -10^9 e 10^9

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 30 pontos, $N \leq 50$ e $M \leq 500$.

Exemplos

Entrada	**Saída**

`6 11`
`34 9 -2 77 -11 5`
`34 5 -2 32 -11 -6 28 66 -2 -22 33`
	`sim`

Entrada	**Saída**
6 8 34 9 -2 77 -11 5 -11 77 -2 75 9 48 7 5	48

Mapa

Nome do arquivo fonte: `mapa.c`, `mapa.cpp`, `mapa.pas`, `mapa.java`, *ou* `mapa.py`

Byteland é uma cidade bastante movimentada, cujo prefeito, Joãozinho, vem lutando recentemente por sua inclusão no grupo das cinco cidades mais importantes de Byteworld. Para uma cidade ser considerada importante em Byteworld, ela precisa seguir alguns critérios. Antes de tudo, vamos definir Byteland, que é uma cidade como qualquer outra, onde esquinas se conectam através de ruas de mão dupla. Sabe-se também que existe um e somente um caminho, sem repetir esquinas, entre qualquer par de esquinas. Além disso, cada rua pode ser considerada importante ou não. Caso ela seja importante, a rua é pintada de branco e caso não seja, é pintada de azul.

Para saber se uma cidade é importante ou não em Byteworld é necessario calcular um valor E: a quantidade de pares de esquinas (A, B) tal que existe ao menos uma rua importante no caminho entre A e B. Note que (A, B) e (B, A) são o mesmo par!

O prefeito de Byteland resolveu pedir sua ajuda para calcular o valor E e saber, assim, se Byteland é ou não uma cidade importante para Byteworld.

Entrada

A primeira linha da entrada contém um inteiro N indicando a quantidade de esquinas em Byteland. As próximas $N-1$ linhas da entrada contêm cada uma três inteiros, A, B e C, indicando que existe uma rua entre as esquinas A e B pintada da cor C. Caso C seja 1, a rua é branca e importante, caso seja 0, a rua é azul e não importante.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o valor E definido acima.

Restrições

- $2 \leq N \leq 10^5$
- $1 \leq A, B \leq N$
- $0 \leq C \leq 1$

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 40 pontos, $N \leq 10^3$.

Exemplos

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

Entrada	**Saída**
6 1 2 0 2 3 1 3 4 0 2 5 0 5 6 1	11

Frequência

Nome do arquivo fonte: `frequencia.c`, `frequencia.cpp`, `frequencia.pas`, `frequencia.java`, *ou* `frequencia.py`

Byteland é uma cidade bastante conhecida por propor variados desafios aos seus habitantes. Recentemente, o prefeito de Byteland, Joãozinho, decidiu propor um desafio que ele gosta de chamar de Tabuleiro da Frequência.

A brincadeira ocorre da seguinte forma. Inicialmente, um tabuleiro com dimensões $N \times N$ é dado contendo apenas 0's. Depois disso, Q operações são propostas, podendo ser de 4 tipos:

- 1 X R: Atribuir o valor R a todos os números da linha X;
- 2 X R: Atribuir o valor R a todos os números da coluna X;
- 3 X: Imprimir o valor mais frequente na linha X;
- 4 X: Imprimir o valor mais frequente da coluna X.

Joãozinho é muito bom com computadores, mas também é bastante preguiçoso. Sabendo que você é um dos melhores programadores do mundo, ele decidiu pedir sua ajuda para resolver este problema.

Entrada

A primeira linha da entrada é composta por dois inteiros N e Q, representando, respectivamente, o tamanho do tabuleiro e a quantidade de operações. As próximas Q linhas da entrada vão conter as Q operações. O primeiro inteiro de cada linha vai indicar o tipo da operação. Caso seja 1 ou 2, será seguido por mais dois inteiros X e R. Caso seja 3 ou 4, será seguido por apenas mais um inteiro X.

Saída

Para cada operação do tipo 3 ou 4, seu programa deve produzir uma linha, contendo o valor da resposta correspondente. Se uma linha ou coluna tiver dois ou mais valores que se repetem o mesmo número de vezes, você deve imprimir o maior deles. Por exemplo, se uma linha tem os valores [5,7,7,2,5,2,1,3], tanto o 2, 5 e 7 se repetem duas vezes, então a resposta será 7, pois é o maior deles.

Restrições

- $1 \leq N, Q \leq 10^5$
- $1 \leq X \leq N$
- $0 \leq R \leq 50$

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 30 pontos, $N \leq 10^3$.
- Em um conjunto de casos de teste equivalente a 20 pontos, apenas as operações 2 e 3 serão usadas.

Exemplos

Entrada	Saída
5 9 3 1 1 1 2 1 3 4 1 4 4 4 2 2 2 5 2 3 5 2 4 5 3 3	0 4 5

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

Entrada	Saída
3 6 1 1 2 1 2 3 1 3 4 4 3 1 3 0 4 3	4 3