OBI2011

Caderno de Tarefas

Modalidade **Programação** • Nível **2**, Fase **1**

19 de março de 2011

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 9 páginas (não contando a folha de rosto), numeradas de 1 a 9. 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 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.

O mar não está para peixe

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

Em um arquipélago no meio do Oceano Pacífico a economia é regida pela pesca, pois o peixe é o principal alimento disponível. Ultimamente, a população desse arquipélago tem aumentado drasticamente, o que levou a um grande aumento da pesca, e, consequentemente, a problemas.

Neste arquipélago, cada pescador vai diariamente ao alto mar com a intenção de conseguir trazer o maior número de peixes para o seu vilarejo. Com a expansão da pesca, os pescadores estão começando a jogar suas redes de pesca por cima das de outros pescadores. Com isso, os pescadores perdem, pois apenas o primeiro pescador pega os peixes da intersecção entre as redes.

A Associação dos Pescadores da ilha decidiu fazer um levantamento para descobrir quanto do mar está de fato sendo aproveitado, ou seja, qual a área do mar que está coberta por pelo menos uma rede de pesca.

Como há muitas intersecções entre as redes de pesca, é muito difícil para a associação calcular a área total da região coberta pelas redes. Por este motivo, eles pediram para que você escrevesse um programa para resolver este problema.

Como é muito difícil navegar pelo mar, os pescadores sempre jogam as redes de forma que as regiões cobertas por cada rede são sempre retângulos com lados paralelos aos eixos, se imaginarmos o mar como um plano cartesiano.

Entrada

A primeira linha da entrada possui um inteiro N indicando o número de redes que foram lançadas. As próximas N linhas descrevem as regiões cobertas pelas redes: cada uma contém quatro inteiros X_i e X_f, Y_i e Y_f. A região coberta pela rede em questão contém todo ponto (X, Y) tal que $X_i \leq X \leq X_f$ e $Y_i \leq Y \leq Y_f$.

Saída

A saída deve conter apenas uma linha contendo a área da região do mar realmente aproveitada pelos pescadores, ou seja, a área total da região do mar coberta por pelo menos uma rede de pesca.

Restrições

- $1 \leq N \leq 100$
- $1 \leq X_i \leq X_f \leq 100$
- $1 \leq Y_i \leq Y_f \leq 100$

Exemplos

Entrada	**Saída**
2 1 3 1 3 2 4 2 4	7

Entrada	**Saída**
3 1 6 1 2 3 7 1 2 2 5 1 2	6

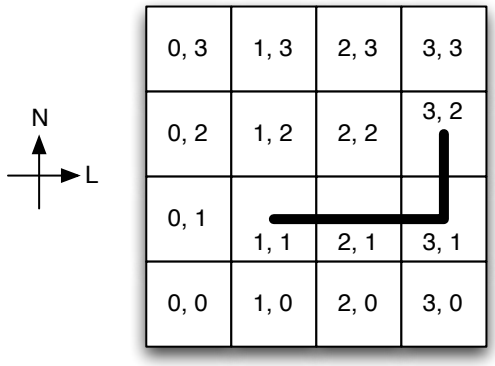
Caça ao Tesouro

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

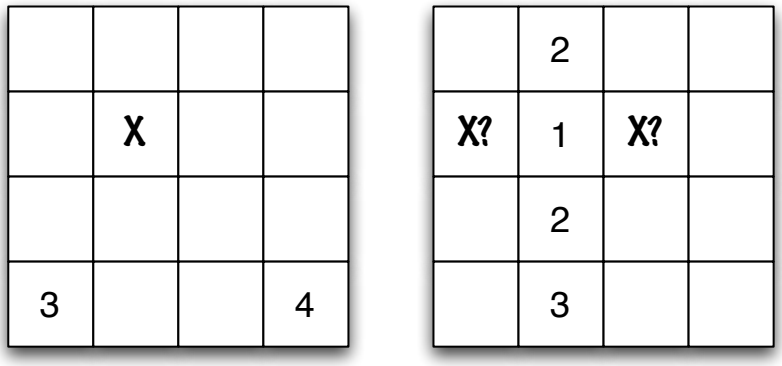
Capitão Tornado é um pirata muito cruel que faz qualquer coisa por dinheiro. Há alguns dias, o capitão soube da existência de um tesouro numa ilha deserta, e agora tenta determinar sua posição.

A ilha pode ser vista como um quadriculado $N \times N$ de terra cuja posição $(0, 0)$ está a sudoeste, a posição $(N - 1, 0)$ está a sudeste, a posição $(0, N - 1)$ está a noroeste e a posição $(N - 1, N - 1)$ está a nordeste. Em alguma posição desse quadriculado está o tesouro.

Uma curiosidade importante é a perna de pau que o capitão possui. Ela impede que o capitão se locomova em direções que não a horizontal ou a vertical: para ir da posição $(1, 1)$ para a posição $(3, 2)$, por exemplo, o capitão é obrigado a gastar três passos. É claro que o capitão sempre escolhe, dentro de suas limitações, um caminho com o menor número de passos possível. Chamamos esse modo de andar de *passos de capitão*. Um exemplo de caminho por *passos de capitão* entre $(1, 1)$ e $(3, 2)$ é ilustrado na figura a seguir.



Como em toda boa caça ao tesouro, o capitão não conhece a posição onde o tesouro se encontra: ele possui um mapa que corresponde à geografia da ilha. Em algumas posições desse mapa, existem pistas escritas. Cada pista consiste em um número D, que indica a menor distância em *passos de capitão* entre a posição em que a pista se encontra e a do tesouro.



Observe que, dependendo da disposição das pistas, a posição do tesouro pode estar determinada de maneira única ou não. Na figura acima e à esquerda, as duas pistas são suficientes para se saber, com certeza, onde está o tesouro; na figura à direita, as quatro pistas dadas ainda possibilitam que tanto a posição $(0, 2)$ quanto a $(2, 2)$ guardem o tesouro. Nesse último caso, não se pode determinar, com certeza, qual é a localização do tesouro.

Dadas as pistas que o capitão possui, sua tarefa é determinar se as pistas fornecem a localização exata do tesouro e, caso positivo, qual ela é.

Entrada

A primeira linha contém dois inteiros positivos N e K, onde N é a dimensão do quadriculado e K é o número de pistas no mapa que o capitão possui.

Cada uma das próximas K linhas contêm três inteiros X, Y e D, informando que existe uma pista na posição (X, Y) contendo o número D. Essa pista indica que o tesouro encontra-se a D *passos de capitão* da posição da pista.

É garantido que, com essas pistas, existe ao menos uma localização possível para o tesouro. Além disso, o mapa não contém duas pistas na mesma posição.

Saída

Se as pistas forem suficientes para determinar com certeza a localização do tesouro, seu programa deve imprimir uma única linha com dois inteiros, X e Y, indicando que o tesouro encontra-se na posição (X, Y).

Caso contrário, seu programa deve imprimir uma única linha com dois inteiros iguais a -1, como nos exemplos de saída a seguir.

Restrições

- $2 \leq N \leq 100$
- $1 \leq K \leq 100$

Exemplos

Entrada	**Saída**
4 2 0 0 3 3 0 4	1 2

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

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

Desafio cartográfico

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

Leonardo Nascimento é um garoto de 13 anos apaixonado por cartografia. Ele assina a lista de discussões da Sociedade Brasileira de Cartografia (SBC) para ficar por dentro de todas as novidades. Em um tópico de discussão na lista da SBC, o presidente da sociedade descobriu que Leonardo tinha apenas 13 anos, e ficou muito feliz em saber que uma pessoa tão jovem tinha tanto interesse pela arte de traçar mapas geográficos e topográficos. Foi então que o presidente resolveu criar desafios com intuito de difundir a cartografia.

Um dos desafios era o seguinte: dado um mapa de cidades ligadas por estradas, determinar a distância entre um par de cidades mais distantes. Como o objetivo era fazer as crianças se divertirem, o presidente resolveu selecionar mapas bem simples. As restrições adotadas foram: (a) todas as estradas são de mão dupla; (b) todas as estradas possuem 1km de comprimento, e portanto toda estrada ligando duas cidades tem o mesmo comprimento; (c) toda estrada conecta apenas duas cidades, e (d) dadas duas cidades quaisquer A e B, só existe uma única maneira de chegar em A partindo de B, e vice-versa.

O presidente da SBC resolveu pedir sua ajuda para escrever um programa de computador que, dado um mapa seguindo as restrições acima, devolva a resposta. Assim, ele conseguirá gerar um gabarito para enviar junto com o desafio.

Entrada

A primeira linha da entrada contém um inteiro N representando o número de cidades no mapa. Cada uma das $N-1$ linhas seguintes da entrada contém dois inteiros A e B indicando que existe uma estrada entre as cidades A e B.

Saída

A única linha da saída contém um inteiro indicando a distância entre um par de cidades mais distantes.

Informações sobre a pontuação

- Em um conjunto de casos de teste que totaliza 20 pontos, $N \leq 200$.
- Em um conjunto de casos de teste que totaliza 40 pontos, $N \leq 1000$.

Restrições

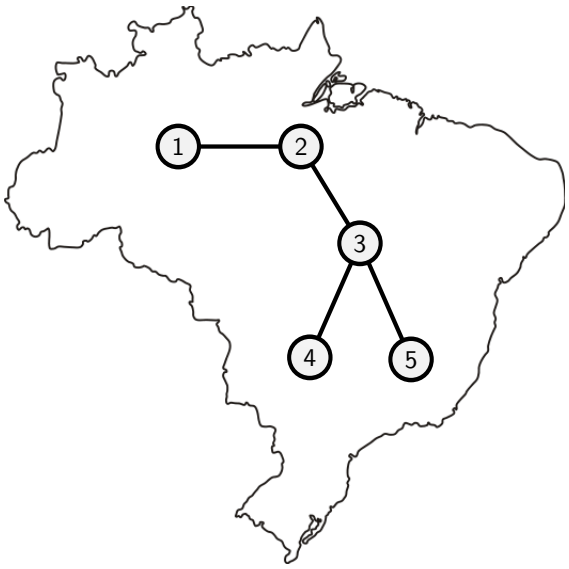
- $2 \leq N \leq 10^6$
- $1 \leq A, B \leq N$ e $A \neq B$

Exemplos

Entrada	**Saída**
10 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10	9

Entrada	**Saída**
5 1 2 2 3 3 4 3 5	3

A figura abaixo ilustra este exemplo, onde temos 5 cidades identificadas por $1, 2, \ldots, 5$. As cidades 1 e 4 estão a uma distância de 3km, assim como as cidades 1 e 5. Não temos nenhum par de cidades que estão a uma distância maior que 3km. Portanto, a resposta para esse caso é 3.



Quadrado Aritmético

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

Arnaldo e Bernardo são dois garotos que compartilham um peculiar gosto por curiosidades matemáticas. Nos últimos tempos, sua principal diversão tem sido investigar propriedades matemágicas de tabuleiros quadrados preenchidos com inteiros. Um belo dia, Arnaldo desenhou o tabuleiro da seguinte figura.

-41	-29	2
28	40	71
11	23	54

— Olha só, várias somas nesse quadrado são iguais! — exclamou Bernardo.

— Como assim? — devolveu, intrigado, Arnaldo.

— Observe:

(-41)	-29	2
28	(40)	71
11	23	(54)

-41	(-29)	2
(28)	40	71
11	23	(54)

-41	(-29)	2
28	40	(71)
(11)	23	54

(-41)	-29	2
28	40	(71)
11	(23)	54

— É mesmo! $(-41) + 40 + 54$ dá 53, $28 + (-29) + 54$ também! — exclamou Arnaldo.

— Eu já verifiquei: existem 6 formas de escolhermos 3 células deste quadrado de forma que cada linha e coluna tenha exatamente uma célula escolhida. Em todas elas, a soma dá 53. Além disso, todos os números são distintos nesse quadrado. — notou Bernardo, exibindo suas habilidades aritméticas.

— Que bacana! Esse quadrado é realmente mágico! Ou, melhor, esse quadrado é realmente *aritmético!* Será que existem mais quadrados como esse?

Uma *escolha legal* de células é uma escolha em que toda linha e toda coluna tenha exatamente uma célula escolhida. Um *quadrado aritmético* de tamanho N e soma S é um tabuleiro de inteiros de N linhas e N colunas em que qualquer *escolha legal* tem soma S e em que todos os números são distintos.

Sua tarefa, neste problema, é gerar um quadrado aritmético de tamanho N e soma S, dados N e S. Como Arnaldo e Bernardo vão querer conferir a sua solução em suas calculadoras, você não deve gerar um quadrado em que alguma célula tenha valor absoluto maior do que 10^9.

Entrada

A primeira e única linha da entrada contém dois números inteiros N e S ($1 \leq N \leq 1000$ e $-1000 \leq S \leq 1000$) representando, respectivamente, o tamanho e a soma do quadrado aritmético pedido.

Saída

Seu programa deve imprimir N linhas, cada uma com N inteiros entre -10^9 e 10^9, representando o quadrado aritmético pedido. Para uma mesma entrada, podem existir vários quadrados aritméticos válidos; seu programa deve imprimir qualquer um deles, mas **apenas um.**

É garantido que existirá pelo menos um quadrado aritmético válido para cada entrada testada.

Restrições

- $1 \leq N \leq 1000$
- $-1000 \leq S \leq 1000$
- $-10^9 \leq$ valor de cada célula $\leq 10^9$

Informações sobre a pontuação

- Em um conjunto de casos de teste que totaliza 30 pontos, $N \leq 3$;
- Em um conjunto de casos de teste que totaliza 70 pontos, $N \leq 100$;

Exemplos

Observe que, neste problema, para uma mesma entrada, podem existir várias saídas corretas. As saídas abaixo indicam apenas uma de várias potenciais soluções. Contanto que sua saída seja um quadrado aritmético válido segundo as condições do enunciado, sua solução será considerada correta, mesmo que não seja igual aos exemplos indicados abaixo.

Entrada	**Saída**
`2 49`	`23 40` `9 26`

Entrada	**Saída**

`3 53`	`-41 -29 2`
	`28 40 71`
	`11 23 54`

Entrada	**Saída**
`1 -55`	`-55`