Competidor(a): _____

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

Este Caderno de Tarefas não pode ser levado para casa após a prova. Após a prova entregue este Caderno de Tarefas para seu professor guardar. Os professores poderão devolver os Cadernos de Tarefas aos competidores após o término do período de aplicação das provas (1 de outubro de 2022).

Olimpíada Brasileira de Informática

OBI2022

Caderno de Tarefas

Modalidade Programação • Nível 1 • Fase 3

1 de outubro de 2022

A PROVA TEM DURAÇÃO DE 4 HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

FUNDAÇÃO BEHRING

Coordenação:

INSTITUTO DE COMPUTAÇÃO UNICAMP

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 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; soluções na linguagem Python 3 devem ser arquivos com sufixo *.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.

Restaurante de pizza

Nome do arquivo: `pizza.c`, `pizza.cpp`, `pizza.pas`, `pizza.java`, `pizza.js` *ou* `pizza.py`

Um amigo seu acabou de se mudar para Linearlandia. Apesar de recém chegado, ele decidiu montar um negócio, um restaurante de Pizza.

Seu amigo está muito feliz na nova empreitada, contudo, com muito medo de errar. Além do preparo da pizza, o restaurante deve se preocupar com a caixa para entrega (que são retangulares), que deverá ser a mesma para todas as pizzas, e com o corte das fatias, que será automatizado e igual para todas as pizzas produzidas.

No momento seu amigo está planejando qual será o tamanho das pizzas (que serão todas círculos perfeitos de um único tamanho), e também qual será o ângulo interno de cada fatia (em graus). Além disso, ele encontrou uma loja de caixas de pizza com ótimos preços, mas não sabe se as caixas são adequadas para as pizzas que ele vai produzir.

A restrição para a caixa de pizza é que a pizza caiba dentro da caixa (mesmo que com alguma folga); a restrição para o ângulo de corte é que todas as fatias sejam de mesmo tamanho (mesmo que seja uma só fatia).

Dados as dimensões da caixa de pizza, o raio da pizza e o ângulo interno da fatia em graus, você deve escrever um programa para determinar se a caixa e o ângulo escolhidos satisfazem às restrições.

Entrada

A entrada é composta por quatro linhas, contendo respectivamente os números inteiros A, B, R e G. Os inteiros A e B são as dimensões da caixa de pizza, o inteiro R é o raio da pizza e o inteiro G é o angulo interno das fatias de pizza.

Saída

Seu programa deve produzir uma única linha na saída, contendo um único caractere, que deve ser `S` se os dados satisfazem às restrições ou `N` caso contrário.

Restrições

- $1 \leq A, B, R \leq 10^9$;
- $1 \leq G \leq 360$.

Informações sobre a pontuação

- Para um conjunto de testes valendo 13 pontos, $A = B$;
- Para um conjunto de testes valendo 26 pontos, $G = 60$;
- Para um conjunto de casos de testes valendo outros 61 pontos, nenhuma restrição adicional.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
4 10 2 60	S

Explicação do exemplo 1: A pizza cabe na caixa e a escolha de ângulo divide a mesma igualmente.

Exemplo de entrada 2	**Exemplo de saída 2**
4 4 3 60	N

Explicação do exemplo 2: O raio da pizza é 3, logo, não cabe em uma caixa de lados de tamanho 4.

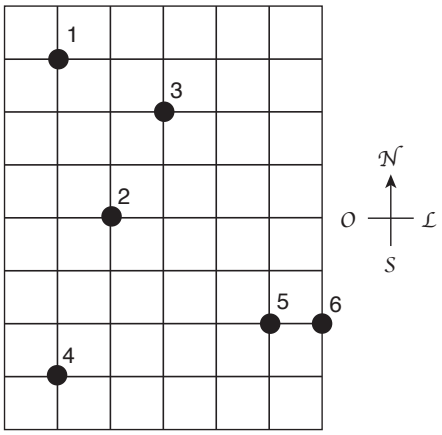
Exemplo de entrada 3	**Exemplo de saída 3**
10 10 5 25	N

Explicação do exemplo 3: O ângulo da fatia é 25, logo, não resulta em fatias iguais.

Carro elétrico

Nome do arquivo: `carro.c`, `carro.cpp`, `carro.pas`, `carro.java`, `carro.js` *ou* `carro.py`

O mapa ao lado mostra o reino de Quadradônia. As estradas são representadas por linhas e as cidades por círculos numerados de 1 a 10. As estradas são igualmente espaçadas com distância de 100 km entre cada par de estradas, sendo orientadas em apenas duas direções: Norte-Sul e Leste-Oeste. Uma empresa de aluguel de carros em Quadradônia utiliza apenas carros elétricos. A *autonomia* de um carro elétrico é a distância que ele pode percorrer com uma carga de energia; após essa distância o carro deve ser carregado novamente para que possa ser utilizado. Há carregadores de energia em cada cidade e não há carregadores de energia fora das cidades. Entre cidades, os carros trafegam apenas pelas estradas e todos os carros têm a mesma autonomia.



Um vendedor deseja partir da cidade 1 e visitar todas as outras cidades, em qualquer ordem, mesmo que ele visite a mesma cidade mais de uma vez. Ele quer utilizar preferencialmente carros elétricos na sua viagem, mas se necessário viajará de avião se a distância para a próxima cidade for maior do que a autonomia do carro. Por exemplo, no mapa acima, se a autonomia for 300 km, o vendedor pode alugar um carro em 1 e visitar 3 e depois 2, mas não pode alcançar as outras cidades. Então ele pode viajar de avião até 5, alugar um carro visitar 6, depois viajar de avião até 4. Assim, são necessárias duas viagens de avião para ele visitar todas as cidades.

Dados o mapa da Quadradônia e a autonomia dos carros, determine qual o menor número de viagens de avião que são necessárias para que o viajante visite todas as cidades, partindo da cidade 1.

Entrada

A primeira linha contém dois inteiros X e Y, indicando respectivamente o número de estradas na direção Oeste-Leste e estradas na direção Norte-Sul. As estradas são numeradas de 1 a X na direção Oeste-Leste e de 1 a Y na direção Norte-Sul. A segunda linha contém dois inteiros N e A, indicando respectivamente o número de cidades e a autonomia dos carros, em kilômetros. As cidades são numeradas de 1 a N. Cada uma das N linhas seguintes contém um par de inteiros x_i e y_i, indicando a posição da cidade de número i, para $1 \leq i \leq N$.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o menor número de viagens de avião necessárias para que o vendedor visite todas as cidades.

Restrições

- $1 \leq X \leq 100\ 000$
- $1 \leq Y \leq 100\ 000$
- $2 \leq N \leq 1\ 000$
- $1 \leq A \leq 150\ 000$
- $1 \leq x_i \leq X$
- $1 \leq y_i \leq Y$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 37 pontos, $N = 2$.
- Para outro conjunto de casos de testes valendo 32 pontos, $Y = 1$, e é garantido que as cidades são dadas em ordem crescente de X (isto é, $x_1 < x_2 < ... < x_n$).
- Para um outro conjunto de casos de testes valendo 31 pontos, nenhuma restrição adicional.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
7 9 6 300 2 2 3 5 4 3 2 8 6 7 7 7	2

Explicação do exemplo 1: Este é o exemplo do enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
7 9 6 200 2 2 3 5 4 3 2 8 6 7 7 7	4

Explicação do exemplo 2: Como a autonomia é 200 km, a única viagem de carro possível é entre as cidades 5 e 6. O vendedor pode por exemplo viajar de avião de 1 para 3, depois de 3 para 2, depois de 2 para 4, depois de 4 para 5, alugar um carro e visitar 6, para um total de quatro viagens de avião.

Dígitos

Nome do arquivo: `digitos.c`, `digitos.cpp`, `digitos.pas`, `digitos.java`, `digitos.js` *ou* `digitos.py`

Joãozinho te propôs o seguinte desafio: ele escolheu dois inteiros A e B, com $1 \leq A \leq B \leq 10^{1000}$, e escreveu na lousa todos os inteiros entre A e B, em sequência, porém colocando um espaço após cada dígito, de forma a não ser possível ver quando um número termina ou começa. Por exemplo, se Joãozinho escolher $A = 98$ e $B = 102$, ele escreveria a sequência "9 8 9 9 1 0 0 1 0 1 1 0 2".

Seu desafio é: dada a lista de dígitos escritos na lousa, encontrar os valores de A e B. Caso exista mais de uma possibilidade para os valores que geraria a lista, você deve encontrar uma em que o valor de A é o menor possível.

É garantido que a lista de dígitos da lousa tem no máximo tamanho 1000.

Entrada

A primeira linha da entrada contém um único inteiro N, indicando o número de dígitos. A segunda linha contém N inteiros d_i, indicando os dígitos escritos.

Saída

Imprima o menor valor possível de A.

Restrições

- $1 \leq N \leq 1000$
- $0 \leq d_i \leq 9$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 21 pontos, $1000 \leq A \leq B \leq 9999$.
- Para outro conjunto de casos de testes valendo 23 pontos, $B = A + 1$.
- Para outro conjunto de casos de testes valendo 40 pontos, $A, B < 10^6$.
- Para outro conjunto de casos de testes valendo 16 pontos, nenhuma restrição adicional.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
6 1 2 3 1 2 4	123

Exemplo de entrada 2	**Exemplo de saída 2**
6 8 9 1 0 1 1	8

Pilhas de moedas

Nome do arquivo: `pilhas.c`, `pilhas.cpp`, `pilhas.pas`, `pilhas.java`, `pilhas.js` *ou* `pilhas.py`

Flávia possui várias moedas em sua coleção, que estão organizadas em N pilhas, cada pilha com um certo número de moedas. Vamos chamar o número de moedas de uma pilha de *altura* da pilha.

A garota pretende adicionar algumas moedas à sua coleção, de forma que cada moeda nova deve ser adicionada em uma das pilhas existentes. As moedas originais, porém, devem permanecer nas suas pilhas.

Flávia está se perguntando agora: qual o número mínimo de moedas que ela deve adicionar à coleção para que, considerando os valores de todas as N novas alturas de pilhas, a quantidade de números distintos seja no máximo K?

Por exemplo, se a lista de alturas inicialmente é $(3, 5, 8, 4, 5, 8)$, temos que existem 4 valores distintos de alturas: 3, 4, 5 e 8. Se $K = 2$, poderíamos, com 3 moedas novas, adicionar duas na pilha de índice 1, e uma na pilha de índice 4. Assim, a lista de alturas ficará $(5, 5, 8, 5, 5, 8)$, que possui apenas dois valores distintos de alturas: 5 e 8.

Note que, se inicialmente a lista de alturas já tem no máximo K valores distintos, Flávia já estaria feliz, e não iria precisar de nenhuma moeda nova.

Entrada

A primeira linha da entrada contém um dois inteiros separados por espaços N, indicando o número de pilhas e K, indicado o número máximo de valores distintos. A segunda linha contém N inteiros P_i, indicando as alturas das pilhas.

Saída

Imprima a menor quantidade adicional de moedas.

Restrições

- $1 \leq N \leq 500$
- $1 \leq K \leq N$
- $1 \leq v_i \leq 500$

Informações sobre a pontuação

- Para um conjunto de casos de testes valendo 13 pontos, $K = 1$.
- Para outro conjunto de casos de testes valendo 21 pontos, $K = 2$.
- Para outro conjunto de casos de testes valendo 28 pontos, $K, N, v_i \leq 50$.
- Para outro conjunto de casos de testes valendo 38 pontos, nenhuma restrição adicional.

Exemplos

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

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