CADERNO DE PROBLEMAS

17/05/2019 – 14h às 17h30

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 6 problemas, sendo que 2 deles estão descritos em inglês.

1. Não é permitido o uso de material impresso, livros, apostilas e dicionários. Apenas é permitido o uso de lápis, caneta, lapiseira, borracha, régua e papel para rascunho (o papel é fornecido pela comissão organizadora). O acesso à internet é bloqueado durante a realização da prova. A ajuda do ambiente de desenvolvimento como C, C++, Java ou Python pode ser utilizada.

2. A correção das resoluções dos problemas será automatizada, por meio do sistema de submissão eletrônica BOCA, tendo como base os resultados obtidos a partir de uma série de execuções dos algoritmos submetidos pelas equipes.

3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e da saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.

4. Os problemas não estão ordenados por ordem de dificuldade neste caderno. A sugestão é de que procure resolver primeiro os problemas mais fáceis.

5. Utilize os nomes indicados nos problemas para nomear os seus arquivos-fonte, de acordo com a linguagem de programação utilizada.

6. Os problemas devem ser resolvidos utilizando o raciocínio entrada-processamento-saída, ou seja, não é necessário armazenar toda a saída para depois exibi-la.

7. Dicas de leitura (entrada de dados) e exibições (saída de dados) encontram-se no verso desta folha.

Faculdade de Informática de Presidente Prudente

http://www.unoeste.br/fipp/maratona

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (tela). Utilize as funções padrão para entrada e saída de dados, como os exemplos a seguir:

- em C: `scanf`, `getchar`, `printf`, `putchar`;
  Para ler um **char** use: `scanf(" %c",&c);` não use: `fflush(stdin); scanf("%c",&c);`

- em C++: as mesmas de C ou os objetos `cin` e `cout`;
  Em C ou C++: use `sprintf(str,"%d",num);` em vez de `itoa(num,str,10);`

  Em C ou C++: não use `include<conio.h>`

- em Java, pode-se usar a classe `Scanner`:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
String s = sc.next();
float f = sc.nextFloat();
```

  Ou ainda as classes `InputStreamReader` e `BufferedReader`:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
try {
    v = br.readLine();
} catch(Exception e){  }
```

  Para exibição:

```
System.out.println();
System.out.printf("%d", i); //também %f, %c, %s, ...
```

  Em Java: não utilize `package`, o arquivo fonte deve a classe com os métodos e o `main`.

- em Python:
  Para leitura de um **int** use: `i = int(input())`
  Para leitura de um **float** use: `f = float(input())`
  Para leitura de um **char** use: `c = input()[0]`
  Para leitura de valores sequencias (quantidade fixa): `10 20 30`

```
v1, v2, v3 = list(map(int, input().split()))
```

  Para leitura de valores sequencias (quantidade não fixa): `10 20 30 ....`

```
valores = list(map(int, input().split()))
for v in valores:
    print(v) # faz alguma coisa com os itens da lista
```

  Para leitura de várias strings (quantidade não fixa): `aaa bbbb ccc`

```
strings = list(map(str, input().split()))
for s in strings:
    print(s) # faz alguma coisa com as strings da lista
```

Unidirecional (vermelha)

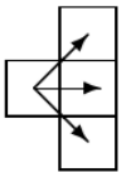
Problema
A

Arquivo fonte: *unidirecional.c*, *unidirecional.cpp*, *unidirecional.java* ou *unidirecional.py*

Problemas que exigem caminhos mínimos para sua solução aparecem em muitas áreas diferentes da ciência da computação. Por exemplo, uma das restrições nos problemas de roteamento do VLSI (*Very-Large-Scale Integration*) é minimizar o comprimento do barramento. O problema do caixeiro viajente – descobrir se todas as cidades na rota de um vendedor podem ser visitadas exatamente uma vez com um limite especificado de tempo de viagem – é um dos exemplos canônicos de um problema NP-completo; as soluções parecem exigir uma quantidade excessiva de tempo para gerar, mas são simples de verificar.

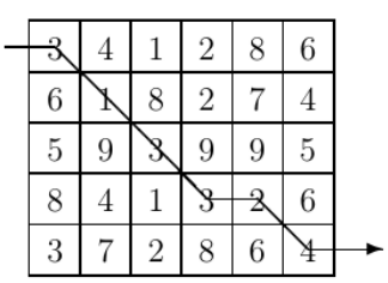
Esse problema tem o objetivo de encontrar um caminho mínimo através de uma grade de pontos, enquanto se percorre o caminho apenas da esquerda para a direita.

Dada uma matriz $m \times n$ de inteiros, você deve escrever um programa que calcule um caminho de peso mínimo. Um caminho começa em qualquer lugar na coluna 1 (a primeira coluna) e consiste em uma sequência de etapas que terminam na coluna n (a última coluna). Um passo consiste em viajar da coluna i para a coluna $i + 1$ em uma linha adjacente (horizontal ou diagonal). A primeira e a última linha (linhas 1 e m) de uma matriz são consideradas adjacentes, de modo que representa um cilindro horizontal. Os passos válidos estão ilustrados à direita.

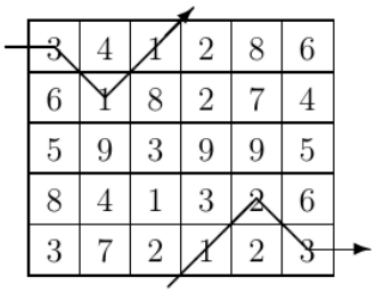


O peso de um caminho é a soma dos inteiros em cada uma das células visitadas da matriz.

Por exemplo, duas matrizes 5×6 ligeiramente diferentes são mostradas abaixo (a única diferença são os números na última linha).



3	4	1	2	8	6
6	1	8	2	7	4
5	9	3	9	9	5
8	4	1	3	2	6
3	7	2	8	6	4



3	4	1	2	8	6
6	1	8	2	7	4
5	9	3	9	9	5
8	4	1	3	2	6
3	7	2	1	2	3

O caminho mínimo é ilustrado para cada matriz. Observe que o caminho da matriz à direita aproveita a propriedade de adjacência da primeira e da última linha.

Entrada

A entrada consiste em uma sequência de especificações de matriz. Cada especificação de matriz consiste nas dimensões de linha e coluna nessa ordem em uma linha, seguidas por $m.n$ inteiros, em que m é a dimensão da linha e n é a dimensão da coluna. Os inteiros aparecem na entrada na ordem sequencial de linha, isto é, os primeiros n inteiros constituem a primeira linha da matriz, os n inteiros seguintes constituem a segunda linha e assim por diante. Os inteiros em uma linha serão

separados de outros inteiros por um espaço. Nota: os números inteiros não estão restritos a serem positivos.

Para cada especificação, o número de linhas será entre 1 e 10 inclusive; o número de colunas será entre 1 e 100 inclusive. Nenhum peso de caminho excederá os valores inteiros representáveis usando 30 bits.

O final da entrada é indicado por um caso em que $m = 0$ e $n = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Duas linhas devem ser impressas para cada especificação de matriz, a primeira linha representa um caminho de peso mínimo e a segunda linha é o custo de um caminho mínimo. O caminho consiste em uma sequência de *n* inteiros (separados por um espaço) representando as linhas que constituem o caminho mínimo. Se houver mais de um caminho de peso mínimo, o caminho que é lexicograficamente menor deve ser produzido.

Nota: Lexicograficamente, significa a ordem natural nas sequências induzidas pela ordem em seus elementos.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
5 6 3 4 1 2 8 6 6 1 8 2 7 4 5 9 3 9 9 5 8 4 1 3 2 6 3 7 2 8 6 4 5 6 3 4 1 2 8 6 6 1 8 2 7 4 5 9 3 9 9 5 8 4 1 3 2 6 3 7 2 1 2 3 2 2 9 10 9 10 0 0	1 2 3 4 4 5 16 1 2 1 5 4 5 11 1 1 19

Exploradores do mundo plano (amarela)

Problema B

Arquivo fonte: *exploradores.c*, *exploradores.cpp*, *exploradores.java* ou *exploradores.py*

Robótica, planejamento de movimento robótico e aprendizado de máquina são áreas que cruzam as fronteiras de muitas das subdisciplinas que compõem a Ciência da Computação: inteligência artificial, algoritmos e complexidade, engenharia elétrica e mecânica, para citar algumas. Além disso, robôs como "tartarugas" (inspirados no trabalho de Papert, Abelson e diSessa) e como "bipadores" (inspirados pelo trabalho de Pattis) foram estudados e usados pelos alunos como uma introdução à programação por muitos anos.

Esse problema envolve determinar a posição de um robô explorando um mundo plano.

Dadas as dimensões de uma grade retangular e uma sequência de posições e instruções do robô, você deve escrever um programa que determine, para cada sequência de posições do robô e instruções, a posição final do robô.

Uma posição do robô consiste em uma coordenada de grade (um par de inteiros: coordenada *x* seguida de coordenada *y*) e uma orientação (N, S, E, W para norte, sul, leste e oeste). Uma instrução de robô é uma sequência das letras "L", "R" e "F" que representam, respectivamente, as instruções:

• Esquerda: o robô vira à esquerda 90 graus e permanece no ponto da grade atual.

• Direita: o robô vira à direita 90 graus e permanece no ponto da grade atual.

• Avanço: o robô avança um ponto da grade na direção da orientação atual e mantém a mesma orientação.

A direção Norte corresponde à direção do ponto da grade (*x*, *y*) ao ponto da grade (*x*, *y* + 1). Como a grade é retangular e limitada, um robô que se move "fora" de uma borda da grade é perdido para sempre. No entanto, os robôs perdidos deixam um "cheiro" robótico que proíbe os futuros robôs de deixar o mundo no mesmo ponto de grade. O cheiro é deixado na última posição da grade ocupada pelo robô antes de desaparecer sobre a borda. Uma instrução para "desligar" o mundo de um ponto de grade do qual um robô foi anteriormente perdido é simplesmente ignorada pelo robô atual.

Entrada

A primeira linha de entrada é a coordenada superior direita do mundo retangular, as coordenadas inferiores esquerda são consideradas como 0, 0.

A entrada restante consiste em uma sequência de posições e instruções do robô (duas linhas por robô). Uma posição consiste em dois inteiros, especificando as coordenadas iniciais do robô e uma orientação (N, S, E, W), todas separadas por espaços em branco em uma linha. Uma instrução de robô é uma string das letras "L", "R" e "F" em uma linha.

Cada robô é processado sequencialmente, ou seja, termina a execução das instruções do robô antes que o próximo robô inicie a execução.

Você pode assumir que todas as posições iniciais do robô estão dentro dos limites da grade especificada. O valor máximo para qualquer coordenada é 50. Todas as sequências de instruções terão menos de 100 caracteres de comprimento.

O final da entrada é indicado por um caso em que as posições iniciais do robô são iguais a zero e a orientação é igual a “X”.

A entrada deve ser lida da entrada padrão.

Saída

Para cada posição/instrução do robô na entrada, a saída deve indicar a posição final da grade e a orientação do robô. Se um robô cair para fora da borda da grade, a palavra “LOST” deve ser impressa após sua posição e orientação.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
5 3 1 1 E RFRFRFRF 3 2 N FRRFLLFFRRFLL 0 3 W LLFFFLFLFL 0 0 X	1 1 E 3 3 N LOST 2 3 S

Heptadecimal Numbers (azul)

Problema C

Arquivo fonte: *heptadecimal.c, heptadecimal.cpp, heptadecimal.java* ou *heptadecimal.py*

The Factory of Computer Enhanced Numbers (FCEN) has asked its Development Committee (DC) to come up with a way to handle numbers written in base 17. As everybody knows, base 17 is very important for many scientific applications, as well as for engineering and other practical uses. Numbers in base 17 can be tough, but are kind and soft if treated appropriately.

Numbers in base 17 are written by using a set of 17 characters: digits 0 to 9 with their usual values, and uppercase letters A to G that have values from 10 to 16, respectively. Base 17, probably because its basement on a prime number, does not require numbers to start with a non-zero digit, so each number has many representations. For instance, the decimal number 117 can be written as 6F, but also as 06F or even 00000006F. Because of this leading-zeroes thing, heptadecimal numbers are hard to compare.

As a member of the FCEN-DC, you were asked to write a program that helps in this difficult and challenging task.

Input

The input contains several test cases. Each test case is described in a single line that contains two non-empty strings of at most 100 heptadecimal digits, separated by a single space. The last line of the input contains two asterisks separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with the sign "<" if the first heptadecimal number is smaller than the second one, the sign ">" if the first heptadecimal number is greater than the second one, or the sign "=" if both heptadecimal numbers are equal.

Please see the sample input and sample output for exact format.

The output must be written to standard output.

Sample input	**Output for the sample input**
006F B3B 0000 0 * *	< =

Tautograms (verde)

Problema D

Arquivo fonte: *tautograms.c*, *tautograms.cpp*, *tautograms.java* ou *tautograms.py*

Fiona has always loved poetry, and recently she discovered a fascinating poetical form. Tautograms area special case of alliteration, which is the occurrence of the same letter at the beginning of adjacent words. In particular, a sentence is a tautogram if all of its words start with the same letter.

For instance, the following sentences are tautograms:

• Flowers Flourish from France

• Sam Simmonds speaks softly

• Peter pIckEd pePPers

• truly tautograms triumph

Fiona wants to dazzle her boyfriend with a romantic letter full of this kind of sentences. Please help Fiona to check if each sentence she wrote down is a tautogram or not.

Input

Each test case is given in a single line that contains a sentence. A sentence consists of a sequence of at most 50 words separated by single spaces. A word is a sequence of at most 20 contiguous uppercase and lowercase letters from the English alphabet. A word contains at least one letter and a sentence contains at least one word.

The last test case is followed by a line containing only a single character '*' (asterisk).

The input must be read from standard input.

Output

For each test case output a single line containing an uppercase 'Y' if the sentence is a tautogram, or an uppercase 'N' otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**

`Flowers Flourish from France`
`Sam Simmonds speaks softly`
`Peter pIckEd pePPers`
`truly tautograms triumph`
`this is NOT a tautogram`
`*`
	`Y`
	`Y`
	`Y`
	`Y`
	`N`

Cadeias Numéricas **(rosa)**

Problema
E

Arquivo fonte: *cadeias.c*, *cadeias.cpp*, *cadeias.java* ou *cadeias.py*

Dado um número, podemos formar uma cadeia numérica:

1. organizando seus dígitos em ordem decrescente;

2. organizando seus dígitos em ordem crescente;

3. subtraindo o número obtido em (2) do número obtido em (1) para formar um novo número;

4. repetindo essas etapas, a menos que o novo número já tenha aparecido na cadeia numérica.

Observe que 0 é um dígito permitido. A quantidade de números distintos na cadeia é o comprimento dela. Você deve escrever um programa que leia números e produza a cadeia numérica e o comprimento dessa cadeia para cada número lido.

Entrada

A entrada consiste em uma sequência de números positivos, todas com menos de 10 dígitos em cada linha.

O final da entrada é indicado quando a cadeia é igual a 0.

A entrada deve ser lida da entrada padrão.

Saída

A saída consiste nas cadeias numéricas geradas pelos números de entrada, seguidas por seus comprimentos, exatamente no formato indicado a seguir. Após cada cadeia numérica e comprimento da cadeia, incluindo o último, deve haver uma linha em branco.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
123456789 1234 444 0	Original number was 123456789 987654321 - 123456789 = 864197532 987654321 - 123456789 = 864197532 Chain length 2 Original number was 1234 4321 - 1234 = 3087 8730 - 378 = 8352 8532 - 2358 = 6174 7641 - 1467 = 6174 Chain length 4 Original number was 444 444 - 444 = 0 0 - 0 = 0 Chain length 2

Pontos em Figuras: Retângulos (branca)

Problema F

Arquivo fonte: *pontos.c*, *pontos.cpp*, *pontos.java* ou *pontos.py*

Dada uma lista de retângulos e uma lista de pontos no plano *x-y*, determine para cada ponto quais figuras (se houver) contêm o ponto.

Entrada

Haverá *n* (≤10) descrições de retângulos, uma por linha. O primeiro caractere designará o tipo de figura ("r" para retângulo). Este caractere será seguido por quatro valores reais designando as coordenadas *x-y* dos cantos superior esquerdo e inferior direito.

O final da lista será sinalizado por uma linha contendo um asterisco na coluna um.

As linhas restantes conterão as coordenadas *x-y*, uma por linha, dos pontos a serem testados. O final desta lista será indicado por um ponto com as coordenadas 9999.9 9999.9; esses valores não devem ser incluídos na saída.

Pontos que coincidam com uma borda de figura não são considerados dentro da figura.

A entrada deve ser lida da entrada padrão.

Saída

Para cada ponto a ser testado, escreva uma mensagem da seguinte forma:

`Point` *`i`* `is contained in figure` *`j`*

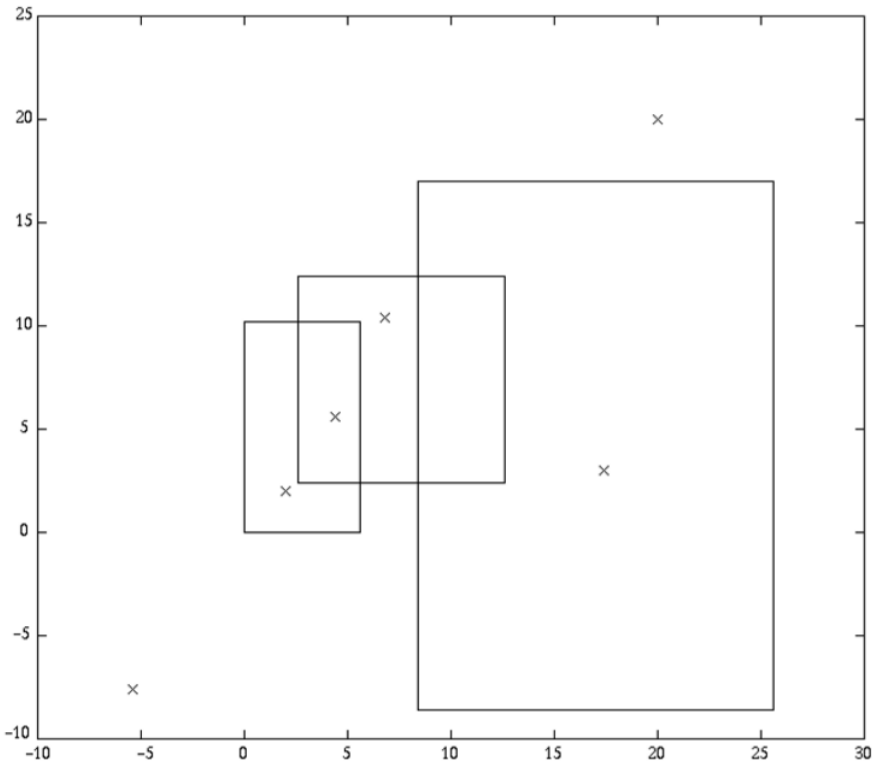
para cada figura que contém o ponto.

Se o ponto não estiver contido em qualquer figura, escreva uma mensagem da seguinte forma:

`Point` *`i`* `is not contained in any figure`

Os pontos e figuras devem ser numerados na ordem em que aparecem na entrada.

Nota: Veja a imagem a seguir para um diagrama dessas figuras e pontos de dados.



A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`r 8.5 17.0 25.5 -8.5`
`r 0.0 10.3 5.5 0.0`
`r 2.5 12.5 12.5 2.5`
`*`
`2.0 2.0`
`4.7 5.3`
`6.9 11.2`
`20.0 20.0`
`17.6 3.2`
`-5.2 -7.8`
`9999.9 9999.9`
	`Point 1 is contained in figure 2`
	`Point 2 is contained in figure 2`
	`Point 2 is contained in figure 3`
	`Point 3 is contained in figure 3`
	`Point 4 is not contained in any figure`
	`Point 5 is contained in figure 1`
	`Point 6 is not contained in any figure`