CADERNO DE PROBLEMAS

25/10/2023 – 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 pelos times.

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.

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 ter 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
```

Macaco-prego (vermelha)

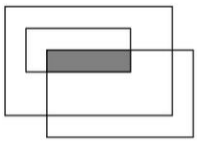
Problema
A

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

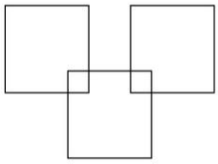
O macaco-prego é um animal irrequieto e barulhento, merecedor também dos adjetivos desordeiro e despudorado. A sua cabeça, encimada por uma densa pelagem negra ou marrom-escura, semelhante a um gorro, torna seu aspecto inconfundível. Apesar de ser o macaco mais comum nas matas do país, uma de suas subespécies encontra-se seriamente ameaçada de extinção: o macaco-prego-do-peito-amarelo, que se distingue das demais pela coloração amarelada do peito e da parte anterior dos braços.

Um grande esforço foi feito pelos primatologistas para aumentar a população dos macacos-prego-do-peito-amarelo. Sabe-se que eles se alimentam de plantas, das quais consomem preferencialmente frutos e brotos. Alimentam-se também de muitos animais, preferencialmente lesmas, lagartas e rãs, e preferem as florestas mais densas. Para determinar o melhor local do país para criar uma nova reserva ambiental para os macacos-prego-do-peito-amarelo, o governo fez um levantamento das regiões no país onde as condições preferidas desses animais ocorrem: regiões de floresta densa, regiões com frutos, regiões com muitos brotos, etc. Ajude a salvar os macacos-prego-do-peito-amarelo.

As regiões propícias para o macaco-prego-do-peito-amarelo foram determinadas como retângulos cujos lados são todos verticais ou horizontais. Sua tarefa é encontrar o local ideal para a reserva ambiental, definida como a interseção de todas as regiões dadas.



Conjunto de três regiões com interseção preenchida em cinza



Conjunto de três regiões com interseção vazia

As regiões foram divididas de tal forma que uma região não tangencia qualquer outra região. Assim, a interseção entre quaisquer duas regiões ou é um retângulo ou é vazia.

Entrada

Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro não negativo, N ($0 \leq N \leq 10000$), que indica o número de regiões (o valor $N = 0$ indica o final da entrada). Seguem-se N linhas, cada uma contendo quatro números inteiros X, Y, U e V ($-10000 \leq X, Y, U, V \leq 10000$) que descrevem uma região: o par X, Y representa a coordenada do canto superior esquerdo e o par U, V representa a coordenada do canto inferior direito de um retângulo.

Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato “Teste n”, onde

n é numerado a partir de 1. A segunda linha deve conter as coordenadas do retângulo de interseção encontrado pelo seu programa, no mesmo formato utilizado na entrada. Caso a interseção seja vazia, a segunda linha deve conter a expressão “nenhum”. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 0 6 8 1 1 5 6 3 2 4 9 0 3 0 4 4 0 3 1 7 -3 6 4 10 0 0	Teste 1 2 4 6 3 Teste 2 nenhum

Máquina de café **(amarela)**

Problema
B

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

O novo prédio da Sociedade Brasileira de Computação (SBC) possui 3 andares. Em determinadas épocas do ano, os funcionários da SBC bebem muito café. Por conta disso, a presidência da SBC decidiu presentear os funcionários com uma nova máquina de expresso. Esta máquina deve ser instalada em um dos 3 andares, mas a instalação deve ser feita de forma que as pessoas não percam muito tempo subindo e descendo escadas.

Cada funcionário da SBC bebe 1 café expresso por dia. Ele precisa ir do andar onde trabalha até o andar onde está a máquina e voltar para seu posto de trabalho. Todo funcionário leva 1 minuto para subir ou descer um andar. Como a SBC se importa muito com a eficiência, ela quer posicionar a máquina de forma a minimizar o tempo total gasto subindo e descendo escadas.

Sua tarefa é ajudar a diretoria a posicionar a máquina de forma a minimizar o tempo total gasto pelos funcionários subindo e descendo escadas.

Entrada

A entrada consiste em 3 números, A_1 , A_2 , A_3 ($0 \leq A_1$, A_2 , $A_3 \leq 1000$), um por linha, onde A_i representa o número de pessoas que trabalham no i-ésimo andar.

Saída

Seu programa deve imprimir uma única linha, contendo o número total de minutos a serem gastos com o melhor posicionamento possível da máquina.

Exemplo de entrada 1	**Saída para o exemplo de entrada 1**
10 20 30	80
Exemplo de entrada 2	**Saída para o exemplo de entrada 2**
10 30 20	60
Exemplo de entrada 3	**Saída para o exemplo de entrada 3**
30 10 20	100

Hamming **(branca)**

Problema
C

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

The Hamming distance between two strings *A* and *B* of equal length is the number of positions where the strings differ. You receive *n* bit sequences, each with length *k*, and your task is to calculate the minimum Hamming distance between two strings.

Input

The first input line has two integers $1 < n < 10^3$ e $0 < k < 31$: the number *n* of bit sequences and their length. Then there are *n* lines, each consisting of a sequence of bits of length *k*.

Output

Print the minimum Hamming distance between two strings.

Sample input	**Output for the sample input**
5 6 110111 001000 100001 101000 101110	1

Seesaw (verde)

Problema D

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

Joãozinho has just changed schools and the first thing he noticed at the new school is that the playground seesaw is not symmetrical, one end is longer than the other. After playing a few times with a friend of the same weight, he realized that when he is at one end, the seesaw is unbalanced to his side (that is, he is at the bottom, and his friend is at the top), but when they switch sides, the seesaw goes unbalanced towards the friend's side. Without understanding the situation, Joãozinho asked for help from another friend from another grade, who explained that the length of the side interferes with the balance of the seesaw, as the seesaw will be balanced when

$$P_1 * C_1 = P_2 * C_2$$

where P_1 and P_2 are the weights of the child on the left and right side, respectively, and C_1 and C_2 are the lengths of the seesaw on the left and right side, respectively.

With the equation, Joãozinho can now tell whether the seesaw is balanced or not but, in addition, he wants to know which way the seesaw will go down if it is unbalanced.

Input

The first and only line of the input contains four integers, P_1, C_1, P_2 and C_2, in this order.

Output

Your program must produce a single line, containing a single integer. If the seesaw is balanced, print "0". If it is unbalanced so that the left child is at the bottom, print "-1", otherwise print "1".

Restrictions

- $10 \leq P_1 \leq 100$ and $10 \leq C_1 \leq 100$
- $10 \leq P_2 \leq 100$ and $10 \leq C_2 \leq 100$

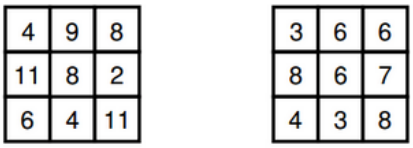
Sample input 1	**Output for the sample input 1**
`30 100 60 50`	`0`
Sample input 2	**Output for the sample input 2**
`40 40 38 60`	`1`

Quadrado (azul)

Problema
E

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

Um quadrado quase mágico, de dimensões $N \times N$, é um quadrado que obedece à seguinte condição. Existe um número inteiro positivo M tal que: para qualquer linha, a soma dos números da linha é igual a M; e para qualquer coluna, a soma dos números da coluna é também igual a M. O quadrado seria mágico, e não apenas quase mágico, se a soma das diagonais também fosse M. Por exemplo, a figura abaixo, parte (a), apresenta um quadrado quase mágico onde $M = 21$.



(a) (b)

Laura construiu um quadrado quase mágico e alterou, propositalmente, um dos números! Nesta tarefa, você deve escrever um programa que, dado o quadrado quase mágico alterado por Laura, descubra qual era o número original antes da alteração e qual número foi colocado no lugar. Por exemplo, na parte (b) da figura, o número original era 1, que Laura alterou para 7.

Entrada

A primeira linha da entrada contém apenas um número N ($3 \leq N \leq 50$), representando a dimensão do quadrado. As N linhas seguintes contêm, cada uma, N números inteiros (entre 1 e 10000), definindo o quadrado. A entrada é garantidamente um quadrado quase mágico onde exatamente um número foi alterado.

Saída

Seu programa deve imprimir apenas uma linha contendo dois números: primeiro o número original e depois o número que Laura colocou no seu lugar.

Exemplo de entrada 1	**Saída para o exemplo de entrada 1**
3 3 6 6 8 6 7 4 3 8	1 7
Exemplo de entrada 2	**Saída para o exemplo de entrada 2**
4 16 3 2 13 5 10 11 8 8 6 7 12 4 15 14 1	9 8

Pastas (rosa)

Problema F

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

Estela é uma secretária dedicada da OBI (Organização Burocrática Internacional), um megaconglomerado empresarial voltado a criação de documentos e preenchimento de formulários. Todo dia ela recebe milhares de pastas suspensas e seu objetivo é organizá-las de uma forma que seja simples recuperar uma pasta do arquivo.

Cada pasta possui uma pequena aba, que fica anexada à pasta e é visível quando a pasta está suspensa em seu arquivo. Todo funcionário fixa a aba em uma das posições especificadas pelo manual de fixação de abas, embora ele possa escolher, ao acaso, qualquer uma das posições descritas no manual. Tais posições são numeradas de 1 até P.

Estela notou que fica consideravelmente mais fácil encontrar as pastas se elas forem arquivadas da seguinte forma: primeiro uma pasta com aba na posição 1, depois uma com aba na posição 2, e assim sucessivamente, até que uma pasta com aba na posição P seja arquivada. Logo após, repete-se o processo, arquivando uma pasta com aba na posição 1. Para Estela, um conjunto de pastas é arquivado de forma perfeita se todas as pastas desse conjunto forem arquivadas da forma descrita anteriormente, ou seja:

- Imediatamente após toda pasta com aba na posição I, $I < P$, existe uma pasta com aba na posição $I + 1$ ou não há nenhuma pasta.
- Imediatamente após toda pasta com aba na posição P, existe uma pasta com aba na posição 1 ou não há nenhuma pasta.
- Todas as pastas do conjunto são armazenadas.

Tarefa

Dado um conjunto de pastas e a posição de suas abas, determinar se é possível arquivar esse conjunto de pastas de forma perfeita.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém dois inteiros P e N que indicam, respectivamente, o número de posições possíveis para se colar as abas ($1 \leq P \leq 1.000$), o número de pastas a serem armazenadas ($1 \leq N \leq 1.000.000$). As N linhas seguintes contém um inteiro I ($1 \leq I \leq P$) cada representando a posição onde a aba da I-ésima pasta foi colada.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo a letra "S" se for possível fazer um arquivamento perfeito ou "N" caso contrário.

Exemplo de entrada 1	Saída para o exemplo de entrada 1
2 2 1 2	S
Exemplo de entrada 2	**Saída para o exemplo de entrada 2**
3 6 1 2 3 1 2 1	N
Exemplo de entrada 3	**Saída para o exemplo de entrada 3**
4 7 1 1 2 2 3 3 4	S