OBI2016

Caderno de Tarefas

Modalidade **Programação** • **Nível Júnior** • Fase **2**

27 de agosto de 2016

A PROVA TEM DURAÇÃO DE **3** HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

v1.0

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 6 páginas (não contando a folha de rosto), numeradas de 1 a 6. 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 instalado e disponível no computador em que você estiver realizando a prova..
- 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; soluções na linguagem Python devem ser arquivos com sufixo *.py2* ou *.py3*, dependendo da versão utilizada; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*. 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 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,raw_ input (apenas Python2), print,write*
  - em Javascript: *scanf, printf*
- 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.

Medalhas

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

A natação foi um dos esportes mais emocionantes das Olimpíadas do Rio. Houve até uma prova na qual três atletas chegaram empatados, cada um recebendo uma medalha de prata! Normalmente, porém, os três primeiros colocados terminam a prova em tempos distintos e, portanto, temos a distribuição mais comum de medalhas: o nadador que terminou no menor tempo recebe medalha de ouro; o nadador que terminou com o segundo menor tempo recebe medalha de prata; e o que terminou com o terceiro menor tempo recebe medalha de bronze. Neste problema, dados os três tempos distintos de finalização da prova, dos três nadadores que ganharam medalhas, seu programa deve dizer quem ganhou medalha de ouro, quem ganhou prata e quem ganhou bronze.

Entrada

A primeira linha da entrada contém um inteiro T_1, indicando o tempo em que o nadador 1 terminou a prova. A segunda linha da entrada contém um inteiro T_2, indicando o tempo de finalização do nadador 2. Por fim, a terceira linha da entrada contém um inteiro T_3, indicando o tempo em que o nadador 3 terminou a prova.

Saída

Seu programa deve imprimir três linhas na saída. A primeira linha deve conter o número do nadador que ganhou medalha de ouro; a segunda linha, o número do nadador que ganhou prata; e a terceira linha, o número do nadador que levou bronze.

Restrições

- Os tempos T_1, T_2 e T_3 são inteiros distintos, com valores entre 1 e 1000, inclusive.

Exemplos

Entrada	**Saída**
230 183 234	2 1 3

Entrada	**Saída**
46 47 48	1 2 3

Entrada	**Saída**
11 21 7	3 1 2

Gincana

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

As duas turmas do terceiro ano de sua escola realizam anualmente uma gincana. Nessa gincana, a delegação de cada turma é dividida em grupos de K pessoas, de forma que K seja o maior número possível que divida as duas delegações sem que sobre alguém. Depois, os grupos competem uns com os outros, ganhando pontos para determinar a turma vencedora. Sua turma pode levar qualquer número X de pessoas entre 1 e M, a quantidade de alunos na turma, e você sabe que a turma rival levará exatamente N pessoas para a gincana. Os integrantes da sua turma são muito bons em competições individuais, mas não trabalham bem em equipe. Portanto, é sua tarefa encontrar a maior delegação possível que sua turma pode levar à competição para que a gincana aconteça com grupos de $K = 1$ pessoa.

Por exemplo, se $N = 9$ e $M = 6$ a sua turma deve levar uma delegação de $X = 5$ pessoas, já que, para esse valor, a única divisão possível é em grupos de $K = 1$ pessoa e, para $X = 6$, os grupos seriam de 3 pessoas.

Entrada

A primeira e única linha contém dois inteiros N e M, representando respectivamente o tamanho da delegação rival e o tamanho da sua turma.

Saída

Seu programa deve produzir uma única linha, contendo um inteiro X, o maior tamanho possível da delegação da sua turma para o qual a gincana aconteça com grupos de uma pessoa.

Restrições

- $1 \leq N, M \leq 10^{18}$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 40 pontos, $N, M \leq 10^3$.
- Em um conjunto de casos de teste equivalente a 60 pontos, $N, M \leq 10^5$.
- Em um conjunto de casos de teste equivalente a 80 pontos, $N, M \leq 10^7$.

Exemplos

Entrada	**Saída**
9 6	5

Entrada	**Saída**
6 9	7

Entrada	**Saída**
6 3	1

Entrada	**Saída**
2310 126	113

Caverna de Ordinskaya

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

Alguns de seus amigos decidiram viajar até a Russia para explorar Ordinskaya, a caverna subaquática mais comprida do país. Apesar da boa visibilidade das águas da caverna sempre é possível encontrar novas passagens e túneis que levam para longe da gruta principal, o que poderia fazer com que alguém se perdesse e provavelmente congelasse nas frias temperaturas observadas ali. Para evitar que algo assim ocorresse durante os mergulhos, o grupo usou uma fita métrica para marcar o caminho feito e garantir um retorno seguro. Além disso aproveitaram para medir quanto haviam explorado, sempre que retornavam à superfície alguém do grupo anotava num caderno o quão longe haviam ido.

O único problema com essa estratégia é que a cada mergulho pessoas diferentes ficavam responsáveis por verificar a fita métrica e anotar quanto havia sido explorado. Assim, se o comprimento da fita era 10 metros, após um mergulho em que o grupo explorou 3 metros da caverna, um dos amigos poderia ter desenrolado a fita do começo para o fim e anotar que 3 metros foram explorados, enquanto outro mais desatento, sem perceber que havia desenrolado a fita no sentido contrário, poderia anotar que 7 metros foram explorados.

Apenas no final da viagem seus amigos perceberam a bagunça feita e agora pediram sua ajuda para reconstruir as distâncias de fato exploradas. Você foi informado que antes da viagem o grupo comprou uma fita com M metros e que no total eles fizeram N mergulhos. Outra informação importante é que a cada novo mergulho pelo menos a mesma distância do mergulho anterior era explorada, então se o comprimento da fita fosse de 10 metros e as anotações feitas fossem 3 e 8 metros, nessa ordem, os únicos cenários que realmente poderiam ter acontecido são:

- 3 metros no primeiro mergulho e 8 no segundo;
- 7 metros no primeiro mergulho e 8 no segundo.

Mas se os valores anotados foram 2 e 8, existem três possibilidades:

- 2 metros no primeiro mergulho e 8 no segundo;
- 2 metros no primeiro mergulho e 2 no segundo;
- 8 metros no primeiro mergulho e 8 no segundo.

Como pode ter ocorrido algum engano nas anotações, pode ser impossível reconstruir a sequência original, não se preocupe, todos vão entender caso isso aconteça.

Entrada

A primeira linha contém dois inteiros N e M, representando respectivamente a quantidade de mergulhos que o grupo fez e o comprimento em metros da fita que levaram para a exploração. A segunda linha contém N inteiros A_1, A_2, ..., A_N representando as medições feitas a cada mergulho, na ordem em que foram anotadas.

Saída

Seu programa deve produzir uma única linha, contendo apenas um inteiro, que representa a soma das distâncias exploradas. Caso exista mais de uma sequência possível, imprima a menor soma das sequencias possíveis. Se não existir nenuma sequência compatível com os dados, imprima apenas o inteiro -1.

Restrições

- $1 \leq N \leq 10^4$.

- $1 \leq M \leq 5 * 10^5$.
- $0 \leq A_i \leq M$.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 20 pontos, $N \leq 20$ e $M \leq 5 * 10^3$.
- Em um conjunto de casos de teste equivalente a 60 pontos, $N \leq 10^3$ e $M \leq 5 * 10^3$.

Exemplos

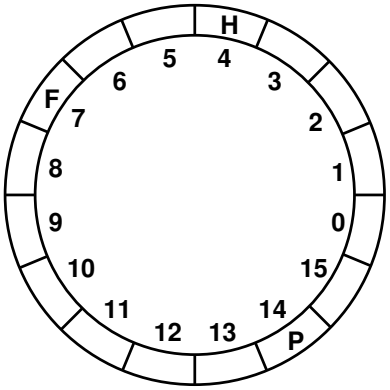
Entrada	**Saída**
`5 7` `2 5 3 6 0`	`20`

Entrada	**Saída**
`3 5` `2 1 2`	`-1`

Fuga com helicóptero

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

Um fugitivo, um helicóptero e um policial estão em posições distintas numa pista circular, exatamente como a mostrada na figura ao lado, com dezesseis posições numeradas de 0 a 15 em direção anti-horária. O helicóptero e o policial ficam sempre parados. O objetivo do fugitivo é chegar no helicóptero sem passar pelo policial antes, claro. Ele pode decidir correr na direção horária, ou na direção anti-horária. Neste problema, dadas as posições do helicóptero, do policial e do fugitivo, e a direção em que o fugitivo decide correr, seu programa deve dizer se ele vai ou não conseguir fugir! Na figura, se o fugitivo decidir correr na direção horária, ele consegue fugir; se decidir correr na direção anti-horária, ele vai ser preso antes de chegar no helicóptero!



Entrada

A entrada consiste de uma linha com quatro inteiros: H, P, F e D, representando, respectivamente, as posições do helicóptero, do policial e do fugitivo, e a direção em que o fugitivo corre, -1 para horário e 1 para anti-horário.

Saída

Seu programa deve imprimir uma linha contendo o caracter "`S`" se o fugitivo consegue fugir, ou "`N`" caso contrário.

Restrições

- Os inteiros H, P e F são distintos e estão entre 0 e 15, inclusive.

Exemplos

Entrada	**Saída**
`4 14 7 -1`	`S`

Entrada	**Saída**
`4 14 7 1`	`N`

Entrada	**Saída**
`15 9 8 -1`	`S`

Entrada	**Saída**
`0 14 15 -1`	`N`