OBI2016

Caderno de Tarefas

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

27 de agosto de 2016

A PROVA TEM DURAÇÃO DE **4** 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 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 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

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 $M \leq 5 * 10^3$.

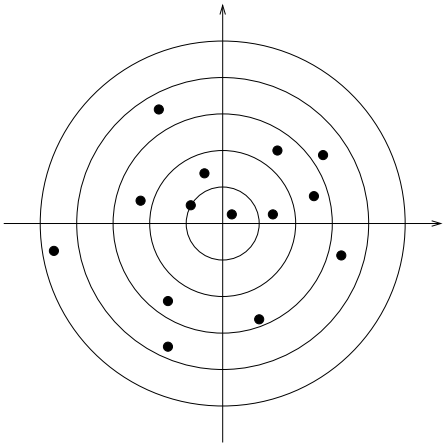
Exemplos

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

Entrada	**Saída**
`3 5` `2 1 2`	`-1`

Arco e flecha

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



O comitê olímpico está testando uma nova forma de pontuar as competições de arco e flecha, baseada em penalidades. O atleta vai atirar N flechas no alvo, em sequência. A penalidade da K-ésima flecha atirada é computada imediatamente após ela atingir o alvo, antes do próximo lançamento, e é igual ao número de flechas que estão no alvo naquele momento cuja distância ao centro do alvo é menor ou igual à distância da K-ésima flecha ao centro, excluindo a própria K-ésima flecha. Quer dizer, a penalidade é o número das $K-1$ flechas lançadas antes da K-ésima flecha que estão mais próximas ou à mesma distância do centro do alvo, comparadas com a K-ésima flecha.
A penalidade total é a soma das penalidades das N flechas. Ganha o atleta que tiver a menor penalidade total ao final. Veja que a penalidade total pode ser zero, se o atleta for bom o bastante para acertar numa sequência estritamente decrescente de distâncias ao centro do alvo.

Neste problema, o centro do alvo está na origem $(0, 0)$. Dada a sequência de coordenadas dos pontos em que as sucessivas flechas atingiram o alvo, seu programa deve computar a penalidade total final do atleta.

Entrada

A primeira linha da entrada contém um inteiro N, representando a quantidade de flechas lançadas. Cada uma das N linhas seguintes contém dois inteiros, X e Y, indicando as coordenadas do ponto em que cada flecha atingiu o alvo, definindo a sequência de lançamentos.

Saída

Imprima uma linha contendo um inteiro representando a penalidade total do atleta.

Restrições

- $1 \leq N \leq 10^5$
- $-10^6 \leq X, Y \leq 10^6$

Informações sobre a pontuação

- Em um conjunto de testes somando 20 pontos, $N \leq 10^4$

Exemplos

Entrada	**Saída**
2 1 3 5 4	1

Entrada	**Saída**
4 -100 85 -25 -60 18 33 0 0	0

Entrada	**Saída**
6 1 1 2 2 2 2 3 3 3 3 3 3	15

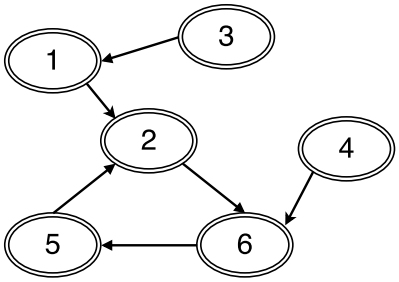
Caminhos do reino

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

O reino de Daglônia é um lugar estranho. Todas as estradas do reino só podem ser usadas em *uma direção* e de cada cidade sai *exatamente uma* estrada. O reino é dividido em duas partes: o *ciclo interno* e os *caminhos periféricos*. Cada uma das cidades do reino pertence a uma dessas partes. No ciclo interno, a estrada que sai de cada cidade vai à próxima cidade do ciclo, de forma que é possível percorrer um caminho que sai de uma cidade qualquer e retorna a essa mesma cidade.

A algumas cidades do ciclo interno pode chegar um dos caminhos periféricos, que são as ligações entre a parte central do reino e o mundo exterior, por onde pessoas podem chegar ao reino (mas não sair). Um caminho periférico começa em uma cidade na qual nenhuma estrada do reino chega e segue pelas estradas de cada cidade até chegar em uma cidade do ciclo interno. A cada cidade pertecente a um caminho periférico chega no máximo uma estrada. A cada cidade do ciclo interno chegam no máximo duas estradas: uma estrada do ciclo interno (que sempre existe) e uma estrada de um caminho periférico (que pode ou não existir).

A figura abaixo mostra um exemplo das cidades e estradas do reino, com cidades numeradas de 1 a N.



Na figura, os caminhos periféricos são $(3 \to 1)$ e (4) e o ciclo interno é $(2 \to 6 \to 5 \to 2)$.

Há rumores de que um país vizinho vai declarar guerra contra a Daglônia, e por isso os habitantes do reino querem se encontrar com seus familiares no menor tempo possível. Você foi contratado pelo Rei para ajudá-las. Você receberá Q perguntas da seguinte forma: dadas as cidades A e B onde estão duas pessoas do reino que querem se encontrar, você deve determinar qual o tempo mínimo em que elas podem se encontrar, considerando que cada estrada é percorrida em uma unidade de tempo. O ponto de encontro das duas pessoas pode ser diferente das cidades iniciais e ambas podem se deslocar *simultaneamente* para chegar ao ponto de encontro.

Considerando o exemplo da figura acima, pessoas nas cidades 4 e 3 podem se encontrar na cidade 2 ou 6 em tempo 3. Pessoas nas cidades 1 e 3 podem se encontrar na cidade 1 em tempo 1. Pessoas nas cidades 6 e 3 podem se encontrar na cidade 2 em tempo 2.

Entrada

A primeira linha contém um inteiro N, representando o número de cidades. As cidades são identificadas por inteiros de 1 a N. A segunda linha contém N inteiros $F_1, F_2, ... F_N$, onde F_i é o destino da estrada que parte da cidade i. A terceira linha contém um inteiro Q, que representa o número de perguntas. As Q linhas seguintes contém dois inteiros A e B, indicando as cidades para as quais você deve responder a pergunta descrita acima. Existe pelo menos um caminho periférico.

Saída

Seu programa deve produzir Q linhas, cada uma contendo um único inteiro, o menor tempo necessário para que as duas pessoas se encontrem em uma cidade qualquer.

Restrições

- $3 \leq N \leq 10^5$
- $1 \leq F_i \leq N$
- $F_i \neq i$
- $1 \leq Q \leq 10^5$
- $1 \leq A, B \leq N$
- O reino representado respeita as condições do enunciado. Particularmente, existe exatamente um ciclo, existe pelo menos uma cidade que não pertence ao ciclo, a cada cidade do ciclo chegam no máximo duas estradas e a cada cidade que não pertence ao ciclo chega no máximo uma estrada.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 40 pontos, $Q = 1$.

Exemplos

Entrada	**Saída**
6 2 6 1 6 2 5 5 4 3 1 3 6 3 5 2 2 2	3 1 2 1 0

Entrada	**Saída**
13 7 3 9 5 3 7 5 2 6 1 10 11 9 10 4 10 10 8 12 11 10 4 7 3 8 11 6 11 3 4 6 12 8 5	3 4 1 3 2 5 3 2 4 2