OBI2017

Caderno de Tarefas

Modalidade **Universitária** • Fase **Nacional**

19 de agosto de 2017

A PROVA TEM DURAÇÃO DE **5** 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 9 páginas (não contando a folha de rosto), numeradas de 1 a 9. 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 2 devem ser arquivos com sufixo *.py2*; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py3*; 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.

Carrinho

Nome do arquivo: `carrinho.c`, `carrinho.cpp`, `carrinho.pas`, `carrinho.java`, `carrinho.js`, `carrinho.py2` *ou* `carrinho.py3`

Um carrinho elétrico, que usa apenas uma bateria com carga inicial de C coulombs, tem uma característica incrível: ele só pode ir à velocidade constante mas pode escolher qualquer velocidade constante, maior do que zero, de V metros por segundo. Só que quanto maior a velocidade, menor a autonomia. Quer dizer, de maneira mais rigorosa, a distância máxima $d_{\max}$ metros que ele pode percorrer é diretamente proporcional à carga inicial da bateria e inversamente proporcional à velocidade: $d_{\max} = \frac{C}{V}$. É incrível mas veja que, mesmo que a carga seja muito pequena, o carrinho sempre pode percorrer qualquer distância, desde que vá a uma velocidade suficientemente pequena!

O carrinho está na posição zero de uma pista reta com comprimento D metros. Há N baterias, com diferentes cargas, colocadas em posições distintas ao longo da pista, uma delas na posição zero. Considere que nosso carrinho ideal consegue fazer um pit-stop instantâneo, trocando de bateria sem perder tempo algum. Ao passar por uma nova bateria ele pode decidir ou não fazer a troca; e ele pode alterar sua velocidade apenas num instante em que troca de bateria. Qual é o tempo mínimo possível para o carrinho chegar ao final da pista?

Entrada

A primeira linha da entrada contém um inteiro N e um real D, respectivamente, o número de baterias e o comprimento da pista. As N linhas seguintes contêm, cada uma, dois reais P e C definindo, respectivamente, a posição e a carga das baterias. Sempre existe uma bateria na posição 0.0 e as baterias são dadas em ordem estritamente crescente de posição.

Saída

Imprima uma linha contendo um real, com exatamente três casas decimais, o tempo mínimo possível em segundos para o carrinho chegar ao final da pista.

Restrições

- $1 \leq N \leq 1000$ e $1.0 \leq D \leq 10000.0$
- $0.0 \leq P < D$ e $0.0 < C < 100.0$

Exemplos

Entrada	**Saída**

`4 10.000`
`0.000 1.000`
`1.200 0.100`
`3.000 10.000`
`7.700 1.000`
	`13.900`

Postes

Nome do arquivo: `postes.c`, `postes.cpp`, `postes.pas`, `postes.java`, `postes.js`, `postes.py2` *ou* `postes.py3`

Seu João é proprietário de uma enorme fazenda, protegida por uma cerca formada por postes de madeira e arame farpado. Cada poste da cerca tem 1 metro de altura. Os postes são colocados separados dois metros um dos outros, ao redor de toda a fazendo, e portanto muitos postes são utilizados.

Infelizmente um incêndio destruiu uma grande parte dos postes da cerca. Alguns postes, mesmo um pouco queimados, ainda podem ser utilizados, desde que sejam reforçados. Outros estão irremediavelmente inutilizados e devem ser substituídos por postes novos.

O engenheiro que trabalha para o Seu João percorreu toda a cerca e fez uma lista dos tamanhos de cada poste depois do incêndio. O engenheiro determinou que, se o poste tem menos do que 50 cm, ele deve ser substituído. Se o poste tem ao menos 50 cm, mas menos do que 85 cm, ele deve ser consertado. Se o poste tem 85 cm ou mais, ele não necessita conserto e pode ser usado normalmente.

Dada a lista com os tamanhos de cada poste, você deve escrever um programa para determinar o número de postes que devem ser substituídos e o número de postes que devem ser reforçados para consertar a cerca da fazenda do Seu João.

Entrada

A primeira linha da entrada contém um inteiro N, indicando o número de postes da cerca. A segunda linha contém N números inteiros X_i, indicando os tamanhos dos postes após o incêndio.

Saída

Seu programa deve produzir uma única linha, contendo dois inteiros: o número de postes que devem ser substituídos, seguido do número de postes que devem ser consertados.

Restrições

- $3 \leq N \leq 1000$
- $0 \leq X_i \leq 100$

Exemplos

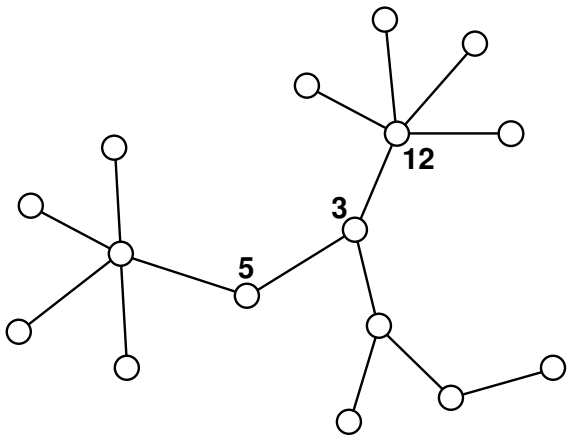
Entrada	**Saída**
5 85 49 50 84 50	1 3

Entrada	**Saída**
4 48 49 30 47	4 0

Dividindo o império

Nome do arquivo: `imperio.c`, `imperio.cpp`, `imperio.pas`, `imperio.java`, `imperio.js`, `imperio.py2` *ou* `imperio.py3`

Um grande Império é composto por N cidades, numeradas de 1 até N. Alguns pares de cidades estão ligados diretamente por estradas bidirecionais e, por uma antiga tradição, esses pares são escolhidos de maneira que sempre é possível ir de qualquer cidade para qualquer outra cidade por *exatamente* um caminho (um caminho é uma sequência de estradas).
O imperador quer dividir seu império em dois para deixar de herança para seus dois filhos. Ele percebeu que basta destruir exatamente uma estrada, qualquer estrada, para dividir seu império em dois menores que, separadamente, preservam a antiga tradição. Ele agora precisa da sua ajuda para computar a menor diferença possível no número de cidades entre os dois impérios resultantes.



Por exemplo, na figura, se o imperador destruir a estrada entre as cidades 3 e 12, os impérios resultantes terão 5 e 11 cidades, uma diferença de 6 cidades. Porém, se ele destruir a estrada entre as cidades 3 e 5, a diferença será de apenas 4 cidades. Você consegue ver que essa é a menor diferença possível para esse exemplo da figura?

Entrada

A primeira linha da entrada contém um inteiro N, representando o número de cidades no império. As $N-1$ linhas seguintes contém, cada uma, dois inteiros A e B, indicando que existe uma estrada bidirecional ligando diretamente as cidades A e B.

Saída

Seu programa deve imprimir uma linha contendo um inteiro representando a menor diferença possível no número de cidades entre os dois impérios resultantes.

Restrições

- $2 \leq N \leq 10^5$
- $1 \leq A \leq N$, $1 \leq B \leq N$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 40 pontos, $N \leq 10000$

Exemplos

Entrada	**Saída**
4 1 2 2 3 3 4	0

Entrada	**Saída**
16 3 5 12 3 5 1 2 1 4 1 6 1 7 1 12 8 12 9 12 10 12 11 3 13 13 14 15 13 15 16	4

Arranha-céu

Nome do arquivo: `arranhaceu.c`, `arranhaceu.cpp`, `arranhaceu.pas`, `arranhaceu.java`, `arranhaceu.js`, `arranhaceu.py2` *ou* `arranhaceu.py3`

Um arranha-céu residencial possui N andares, numerados de 1 a N. O síndico do arranha-céu está tendo muito trabalho com uma nova regra do corpo de bombeiros. Ele não sabe o porquê, mas os bombeiros apontam um andar k e exigem que o síndico informe o total de pessoas que moram no arranha-céu do andar 1 até o andar k, inclusive. Talvez seja alguma medida de segurança dos bombeiros! O problema é que o prédio tem muitos andares e toda hora tem gente se mudando, passando a morar no arranha-céu, ou indo embora. O síndico precisa cuidar de dois eventos:

- *Mudança*: alterar o número de pessoas que moram num determinado andar;
- *Bombeiro*: informar o total de pessoas que moram do andar 1 até um determinado andar, inclusive.

Dados o número de pessoas que moram em cada andar do arranha-céu inicialmente, e uma sequência de eventos (do tipo *Mudança* ou *Bombeiro*), seu programa deve imprimir, para cada evento do tipo *Bombeiro*, o total de pessoas exigido, no momento do evento!

Entrada

A primeira linha da entrada contém dois inteiros N e Q, representando, respectivamente, o número de andares e o número de eventos. A segunda linha contém N inteiros A_i, $1 \leq i \leq N$, indicando o número de pessoas que moram no i-ésimo andar inicialmente. Cada uma das Q linhas seguintes representa um evento e tem uma de duas formas:

- "`0` K P", *Mudança*, alterar o número de pessoas que moram no K-ésimo andar para P pessoas;
- "`1` K", *Bombeiro*, informar o total de pessoas que moram do andar 1 até o andar K, inclusive.

Saída

Para cada evento do tipo *Bombeiro*, seu programa deve imprimir uma linha contendo um inteiro representando o total de pessoas correspondente aquele evento.

Restrições

- $1 \leq N \leq 10^5$ e $1 \leq Q \leq N$
- Há pelo menos um evento do tipo *Bombeiro*
- $1 \leq K \leq N$
- $0 \leq A_i \leq 1000$ e $0 \leq P \leq 1000$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 20 pontos, $N \leq 20000$

Exemplos

Entrada	**Saída**
8 4 30 2 0 42 10 11 11 9 1 5 0 4 12 1 5 1 1	84 54 30

Entrada	**Saída**
1 1 0 1 1	0

Código

Nome do arquivo: `codigo.c`, `codigo.cpp`, `codigo.pas`, `codigo.java`, `codigo.js`, `codigo.py2` *ou* `codigo.py3`

A professora Maryam está tentando construir um código constituído de uma sequência de N strings de 10 letras minúsculas, $S_1, S_2, S_3, \ldots, S_N$. Essas strings da sequência serão, no futuro, concatenadas de diversas maneiras para formar strings maiores. Mas, para que o código seja válido, a sequência de strings tem que satifazer uma propriedade bastante específica: nenhuma string da sequência pode ser substring de uma concatenação de duas strings anteriores na sequência. De forma mais rigorosa, o código será *inválido* se existirem três inteiros a, b e k, tais que:

- $1 \leq a < k \leq N$, $1 \leq b < k \leq N$ (a pode ser igual a b); e
- S_k é substring da concatenação $S_a S_b$.

Por exemplo, o código $S = \{$`aaaaaaabbb`, `yyuudiwwkl`, `kkfidaaooa`$\}$ é válido. Mas se adicionarmos a string `aooaaaaaaa` no final da sequência, o código resultante, $S' = \{$`aaaaaaabbb`, `yyuudiwwkl`, `kkfidaaooa`, `aooaaaaaaa`$\}$, será inválido, pois S'_4 é substring da concatenação $S'_3 S'_1$.

Dada a sequência de strings, seu programa deve determinar se o código é válido, ou não.

Entrada

A primeira linha da entrada contém um inteiro N, representando o número de strings na sequência. As N linhas seguintes contêm, cada uma, uma string de 10 letras minúsculas, definindo a sequência de strings do código.

Saída

Seu programa deve imprimir uma linha contendo a string "`ok`" caso o código seja válido, ou contendo a primeira string na sequência que invalida o código. Quer dizer, contendo S_k onde k é o menor possível tal que S_k seja substring de uma concatenação de duas strings anteriores na sequência.

Restrições

- $1 \leq N \leq 10000$

Informações sobre a pontuação

- Em um conjunto de casos de teste somando 40 pontos, $N \leq 100$

Exemplos

Entrada	**Saída**

`3`
`aaaaaaabbb`
`yyuudiwwkl`
`kkfidaaooa`
	`ok`

Entrada	**Saída**

`4`
`aaaaaaabbb`
`yyuudiwwkl`
`kkfidaaooa`
`aooaaaaaaa`
	`aooaaaaaaa`

Entrada	**Saída**
1 jfjshiddds	ok

Entrada	**Saída**
2 abcdefghij abcdefghij	abcdefghij

Entrada	**Saída**
8 xfwvijuydq hcprvezofg hwykagqawu givfzndqpy yvfiqgadfc wuhcprvezo qaswiksscl uchskpkcit	wuhcprvezo