



OBI2015

Caderno de Tarefas

Modalidade Programação • Nível 1 • Fase 2

29 de agosto de 2015

A PROVA TEM DURAÇÃO DE 4 HORAS

Promoção:



Sociedade Brasileira de Computação

Apoio:



Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 7 páginas (não contando a folha de rosto), numeradas de 1 a 7. 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.
- 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 *.py*; 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, 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.

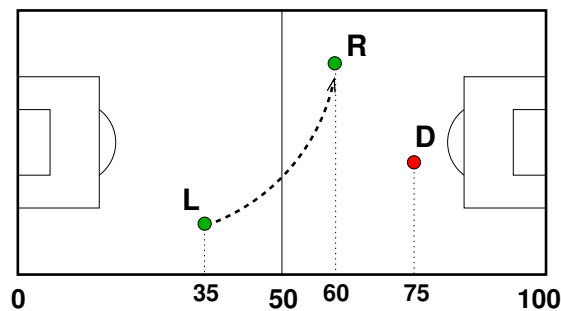
Impedimento!

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

A regra do impedimento no futebol pode parecer estranha, mas sem ela, se a gente pensar bem, o jogo ficaria muito chato! Ela funciona dadas as posições de três jogadores: L o jogador atacante que lança a bola; R o jogador atacante que recebe a bola; e D o último jogador defensor. E a regra vale somente se o jogador R está no seu campo de ataque; se o jogador R está no seu campo de defesa ou na linha divisória do meio campo, ele não está em impedimento. Neste problema o campo tem 100 metros de comprimento. Dadas as posições desses três jogadores, no momento exato do lançamento, haverá impedimento se e somente se a seguinte condição for verdadeira:

$$(R > 50) \text{ e } (L < R) \text{ e } (R > D)$$

A regra parece estranha, não é mesmo? Mas a gente nem precisa entender a lógica dela. O seu programa deve apenas determinar, dadas as três posições L , R e D , se há ou não impedimento, implementando exatamente a condição acima. A figura abaixo mostra um exemplo onde **não** há impedimento:



Entrada

A entrada é composta de apenas uma linha, contendo os três inteiros L , R e D .

Saída

Seu programa deve produzir uma única linha, contendo um único caractere, que deve ser “S” caso haja impedimento, ou “N” caso contrário.

Restrições

- $0 \leq L \leq 100$
- $0 \leq R \leq 100$
- $0 \leq D \leq 100$

Exemplos

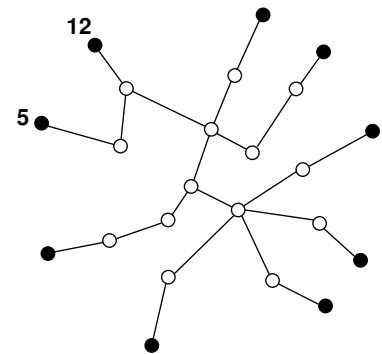
Entrada 35 60 75	Saída N
Entrada 55 68 67	Saída S

Entrada	Saída
66 80 80	N

Capitais

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

A Linearlândia construiu uma rede de ferrovias de alta velocidade, ligando certos pares de cidades, de modo que: é possível viajar entre qualquer par de cidades usando apenas ferrovias; e há apenas um caminho de ferrovias (sequência de ferrovias) entre qualquer par de cidades. Existe muita disputa entre as capitais dos estados da Linearlândia e, por isso, ficou decidido que cada capital seria ligada por ferrovia a apenas uma outra cidade, e que toda cidade que não é capital seria ligada a outras cidades por duas ou mais ferrovias. Dessa forma, nenhuma viagem entre um par de capitais usando apenas ferrovias passa por uma terceira capital.



Vamos definir como *distância-ferrovia* entre duas cidades o número de ferrovias que é necessário utilizar para viajar entre essas duas cidades. Dada apenas a informação sobre quais pares de cidades estão ligados por uma ferrovia, você deve escrever um programa para computar a menor distância-ferrovia entre todos os pares de capitais. Na figura acima, há nove capitais e a menor distância-ferrovia entre qualquer par de capitais é 3, entre as capitais 5 e 12.

Entrada

A primeira linha da entrada contém um inteiro N , o número de cidades. As cidades são identificadas por inteiros de 1 a N . As $N-1$ linhas seguintes contém, cada uma, dois inteiros U e V , representando um par de cidades ligadas por uma ferrovia.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, a menor distância-ferrovia entre todos os pares capitais.

Restrições

- $2 \leq N \leq 10^5$

Informações sobre a pontuação

- Em um conjunto de casos de teste cuja soma é 40 pontos, $N \leq 10^4$;

Exemplos

<p>Entrada</p> <p>8 1 2 2 3 3 4 5 3 6 5 6 7 8 7</p>	<p>Saída</p> <p>3</p>
<p>Entrada</p> <p>2 1 2</p>	<p>Saída</p> <p>1</p>

Letras

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

Uma *cadeia de caracteres* é uma sequência de letras do alfabeto. Uma cadeia de caracteres *crescente* é uma sequência de letras onde a próxima letra (da esquerda para a direita) nunca ocorre antes no alfabeto do que a letra anterior. Por exemplo ABBD é crescente, enquanto ABBAD não é crescente.

Uma *subsequência* de uma cadeia de caracteres é uma cadeia de caracteres que pode ser obtida a partir da remoção de zero ou mais caracteres da cadeia de caracteres original. Por exemplo ANNA é uma subsequência de BANANAS. Outro exemplo seria ANNS, que é uma subsequência crescente de BANANAS.

Dada uma cadeia de caracteres S , escreva um programa para determinar o tamanho da maior subsequência de S que é uma cadeia de caracteres crescente.

Entrada

A entrada consiste em uma única linha, contendo uma cadeia de caracteres S .

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o tamanho da maior subsequência de S que é uma cadeia de caracteres crescente.

Restrições

- A cadeia de caracteres de entrada contém letras maiúsculas do alfabeto, de A até Z.
- $1 \leq \text{comprimento}(S) \leq 3 \times 10^5$.

Informações sobre a pontuação

- Em um conjunto de casos de teste valendo 20 pontos: $\text{comprimento}(S) \leq 20$.
- Em um conjunto de casos de teste valendo 30 pontos: $\text{comprimento}(S) \leq 3000$.


Exemplos

Entrada BANANAS	Saída 4
Entrada AAXBXZZX	Saída 7
Entrada AAA	Saída 3

Torre

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

No jogo de xadrez, a torre é uma peça que pode se mover para qualquer outra posição do tabuleiro na linha ou na coluna da posição que ela ocupa. O professor Paulo está tentando inventar um novo tipo de jogo de xadrez onde todas as peças são torres, o tabuleiro também é quadrado mas pode ter qualquer dimensão e cada posição do tabuleiro é anotada com um número inteiro positivo, como na figura ao lado.

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

Ele definiu o **peso** de uma posição (i, j) como sendo a soma de todos os números que estejam na linha i com todos os números da coluna j , mas sem somar o número que está exatamente na posição (i, j) . Quer dizer, se uma torre estiver na posição (i, j) , o peso da posição é a soma de todas as posições que essa torre poderia atacar.

O professor Paulo está solicitando a sua ajuda para implementar um programa que determine qual é o peso máximo entre todas as posições do tabuleiro. No exemplo da figura acima, com um tabuleiro de dimensão seis (ou seja, seis linhas por seis colunas), o peso máximo é 67, referente à posição (4,4).

Entrada

A primeira linha da entrada contém um inteiro N , representando a dimensão do tabuleiro. Cada uma das N linhas seguintes contém N inteiros positivos X_i , definindo os números em cada posição do tabuleiro.

Saída

Seu programa deve produzir uma única linha, contendo um único inteiro, o peso máximo entre todas as posições do tabuleiro.

Restrições

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

Informações sobre a pontuação

- Em um conjunto de casos de teste cuja soma é 60 pontos, $N \leq 300$.

Exemplos

Entrada	Saída
6 4 1 3 8 4 5 9 2 8 9 2 7 5 5 4 3 2 5 8 2 9 1 9 8 7 1 3 2 1 2 5 1 2 9 3 8	67

Entrada	Saída
3 5 1 1 5 2 1 8 5 5	20