OBI2016

Caderno de Tarefas

Modalidade **Universitária** • Fase **2**

27 de agosto de 2016

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 11 páginas (não contando a folha de rosto), numeradas de 1 a 11. 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.

Pô, que mão

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

Um novo jogo se tornou popular entre jovens de todas as idades recentemente: o "Pô, que mão". Trata-se de um jogo onde uma mão captura criaturas raras e depois as força a lutarem umas contra as outras. Uma verdadeira barbárie.

Ainda assim, o jogo se tornou bastante popular. As criaturas são chamadas de "pô-que-mãos". No jogo, você pode dar doces para as pô-que-mãos, para que elas fiquem mais fortes e evoluam. Como há poucos doces, nem sempre é possível evoluir todas as pô-que-mãos que um jogador possui.

Um jogador tem exatamente 3 pô-que-mãos. Cada um deles necessita de uma quantidade de doces para evoluir. Conhecendo-se a quantidade de doces disponíveis, escreva um programa para determinar qual o maior número de pô-que-mãos que podem evoluir.

Entrada

A entrada é composta por quatro linhas, cada uma contendo um inteiro. A primeira linha contém N, o número de doces disponíveis. A segunda linha contém X, o número de doces necessários para a primeira pô-que-mão evoluir. A próxima linha contém Y, o número de doces necessários para a segunda pô-que-mão evoluir. A última linha contém Z, o número de doces necessários para a terceira pô-que-mão evoluir.

Saída

Seu programa deve produzir uma única linha, contendo um inteiro, o maior número possível de pô-que-mãos que podem evoluir.

Restrições

- $0 \leq N \leq 1000$
- $1 \leq X \leq 1000$
- $1 \leq Y \leq 1000$
- $1 \leq Z \leq 1000$

Exemplos

Entrada	**Saída**
300 220 100 190	2

Entrada	**Saída**
1000 100 200 300	3

Times

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

Chegou a hora da caça ao tesouro na gincana escolar! Para jogar, sua turma de N pessoas deve se dividir em dois times. A divisão será feita com base no número de chamada de cada aluno, um número entre 1 e N. Só que mais importante que fazer uma divisão balanceada é evitar que pessoas que não se gostam fiquem no mesmo time. Você receberá o número N de alunos e, para cada pessoa, você receberá a lista de pessoas que ela não gosta. Se a pessoa A não gosta da pessoa B então a pessoa B também não gosta da pessoa A. O aluno de número 1 sempre ficará no primeiro time. Você deve dividir os outros alunos entre os dois times de forma que dentro de cada time não haja duas pessoas que não se gostam.

Entrada

A primeira linha contém um inteiro N, representando o número de alunos na turma. As N linhas seguintes, para $1 \leq i \leq N$, contém um inteiro M_i, indicando de quantas pessoas o aluno i não gosta, e em seguida M_i inteiros X_j, indicando que a pessoa i não gosta da pessoa X_j.

Saída

Seu programa deve produzir duas linhas, a primeira contendo os números dos integrantes do primeiro time e a segunda contendo os números dos integrantes do segundo time, ambas em ordem crescente. Lembre-se que o aluno 1 sempre estará no primeiro time. Sempre há uma única solução.

Restrições

- $2 \leq N \leq 10^5$.
- $1 \leq M_i \leq N - 1$.
- $1 \leq X_j \leq N$.
- $1 \leq \sum_{i=1}^{N} M_i \leq 6 \times 10^5$.
- Ou seja, a soma de todos os M_i descritos na entrada é no máximo 6×10^5.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 40 pontos, $N \leq 15$.

Exemplos

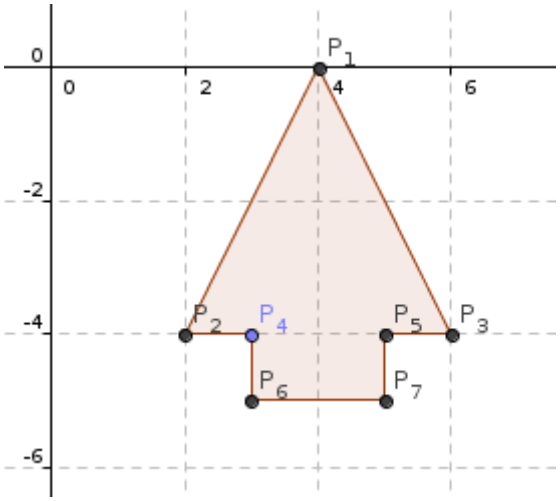
Entrada	**Saída**
13 2 12 9 1 10 3 8 12 13 5 8 10 13 6 5 2 11 4 1 4 1 8 3 4 3 7 1 1 2 2 4 1 5 2 3 1 2 4 3	1 2 3 4 7 11 5 6 8 9 10 12 13

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

Jardim de infância

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

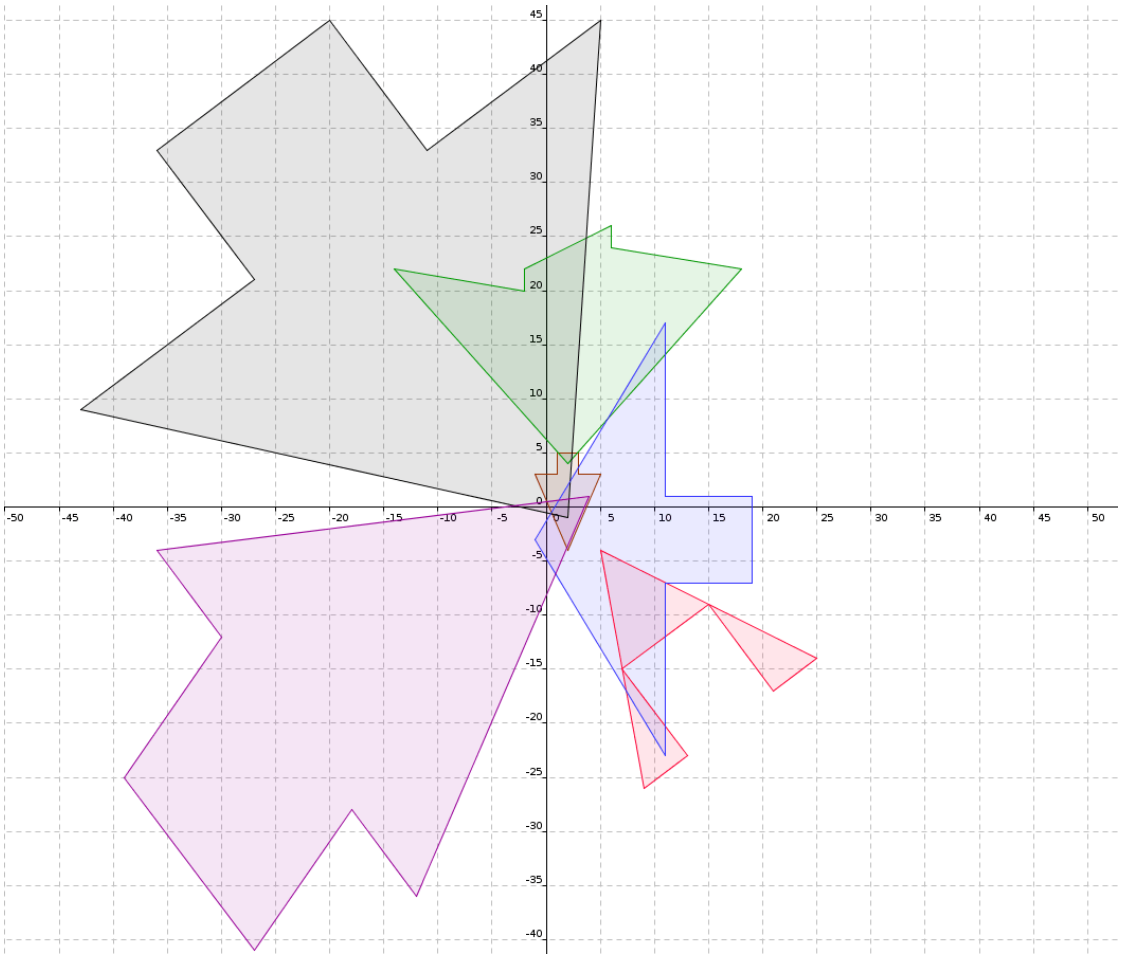
Vívian é uma professora do jardim de infância. Todos os dias, ao final da aula, ela tem que olhar os desenhos que seus alunos fizeram naquele dia e fazer algum comentário. Esta é uma tarefa muito repetitiva, já que as crianças costumam desenhar coisas semelhantes, portanto Vívian decidiu automatizar o processo. Ela fez um programa capaz de processar a imagem e procurar padrões conhecidos para fazer comentários predeterminados. Em particular, ela percebeu que na maioria dos desenhos as crianças incluem um pinheiro. Porém, ela está tendo dificuldades para reconhecê-los e pediu sua ajuda. O programa dela já é capaz de reconhecer uma figura que pode ser um pinheiro e transformá-la em sete pontos $P_1, P_2, \ldots P_7$. O candidato a pinheiro seria a região interna do polígono $P_1P_2P_4P_6P_7P_5P_3$, como mostra a figura a seguir de um pinheiro válido.



Logo, dados os sete pontos que formam a imagem, você deve decidir se ela é ou não um pinheiro. Ao analisar os desenhos das crianças, você decidiu que as condições para que os pontos formem um pinheiro são as seguintes:

- O ângulo $\angle P_2P_1P_3$ é agudo (vértice em P_1);
- Os segmentos $\overline{P_1P_2}$ e $\overline{P_1P_3}$ têm o mesmo comprimento;
- Os pontos P_2, P_3, P_4 e P_5 são colineares;
- Os pontos médios dos segmentos $\overline{P_2P_3}$ e $\overline{P_4P_5}$ são coincidentes;
- O segmento $\overline{P_2P_3}$ tem comprimento maior que o segmento $\overline{P_4P_5}$;
- Os segmentos $\overline{P_4P_6}$ e $\overline{P_5P_7}$ são perpendiculares ao segmento $\overline{P_2P_3}$;
- Os segmentos $\overline{P_4P_6}$ e $\overline{P_5P_7}$ têm o mesmo comprimento;
- Os pontos P_1 e P_6 devem estar separados pela reta que contém o segmento $\overline{P_2P_3}$. Formalmente, o segmento $\overline{P_1P_6}$ deve interceptar a reta que contém o segmento $\overline{P_2P_3}$ em um único ponto.

A imagem a seguir mostra os polígonos formados pelos exemplos de entrada.



Entrada

A entrada contém sete linhas. A i-ésima da entrada contém dois inteiros X_i e Y_i, indicando as coordenadas cartesianas do ponto P_i.

Saída

Seu programa deve produzir uma única linha, contendo uma única letra, `"S"` se os pontos formam um pinheiro pelas condições descritas e `"N"`, caso contrário.

Restrições

- $-2 \times 10^4 \leq X_i, Y_i \leq 2 \times 10^4$.
- Todos os pontos são diferentes.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 50 pontos, o segmento $\overline{P_2P_3}$ será paralelo ao eixo X do plano cartesiano (exemplos 1 e 4).

Exemplos

Entrada	**Saída**
2 -4 5 3 -1 3 3 3 1 3 3 5 1 5	S

Entrada	**Saída**
2 -1 5 45 -43 9 -11 33 -27 21 -20 45 -36 33	S

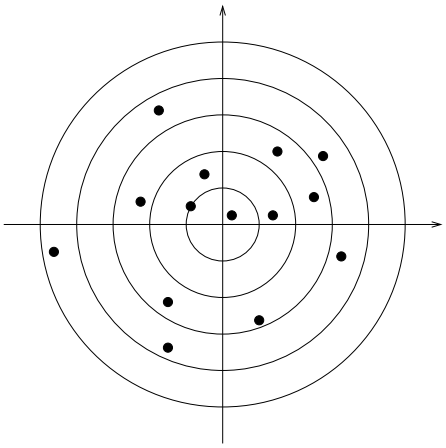
Entrada	**Saída**
-1 -3 11 -23 11 17 11 -7 11 1 19 -7 19 1	N

Entrada	**Saída**
2 4 18 22 -14 22 6 24 -2 20 6 26 -2 22	N

Entrada	**Saída**
4 1 -36 -4 -12 -36 -30 -12 -18 -28 -39 -25 -27 -41	N

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 P_K 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.

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 de cada flecha.

Porém, para dificultar um pouco, você deverá computar a penalidade de cada flecha antes de saber as coordenadas dos pontos posteriores. Para descobrir as coordenadas reais da K-ésima flecha (X^R e Y^R), você deverá somar a penalidade conquistada pela flecha anterior às coordenadas X e Y fornecidas na entrada. Ou seja, $X_K^R = X_K + P_{K-1}$ e $Y_K^R = Y_K + P_{K-1}$. Para a primeira flecha, $X_1^R = X_1$ e $Y_1^R = Y_1$.

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_K e Y_K, indicando as coordenadas que devem ser usadas para calcular o ponto em que cada flecha atingiu o alvo, definindo a sequência de lançamentos.

Saída

Você deve imprimir N linhas. Para $1 \leq K \leq N$, a K-ésima linha deve conter um inteiro P_K representando a penalidade que o atleta rebeceu pela K-ésima flecha.

Restrições

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

Informações sobre a pontuação

- Em um conjunto de testes somando 20 pontos, $N \leq 10^4$
- Em um conjunto de testes somando 60 pontos, $N \leq 5 \times 10^4$

Exemplos

Entrada	**Saída**
2 1 3 5 4	0 1

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

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

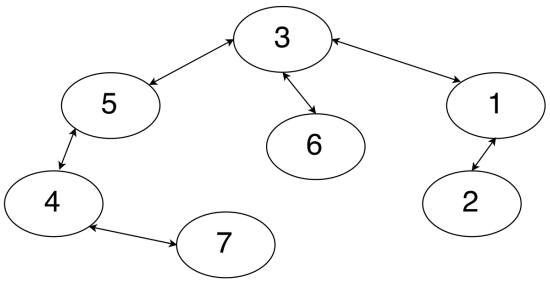
Ciclovias

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

A cidade de Nlogônia é mundialmente conhecida pelas suas iniciativas de preservação ambiental. Dentre elas, uma das que mais chama atenção é a existência de ciclovias em todas as ruas da cidade. Essa medida teve um sucesso tão grande, que agora a maioria dos moradores usa a bicicleta diariamente. Em Nlogônia, as interseções são numeradas de 1 até N. Cada rua liga duas interseções A e B e possui uma ciclovia entre A e B. Um caminho P de tamanho K é definido como uma sequência de interseções $P_1, P_2, \ldots, P_K$, tal que para todo i, $1 \leq i < K$, existe uma ciclovia entre P_i e P_{i+1}. Arnaldo e Bernardo estavam passeando de bicileta pelas ruas de Nlognônia quando pensaram em um novo jogo. Nesse jogo, os dois partem de alguma interseção C e procuram o caminho P de maior tamanho que satisfaça a seguinte regra: as subsequências

$$P_1, P_3, P_5, \ldots, P_{2x+1} \quad \text{e} \quad P_2, P_4, P_6, \ldots, P_{2x}$$

da sequência P devem ser ambas crescentes. Ganha o jogo aquele que encontrar o maior caminho. Bernardo te ligou pedindo ajuda para se preparar para o jogo. Com o mapa da cidade você deve encontrar o tamanho do maior caminho possível para todas as interseções iniciais possíveis, seguindo as restrições acima. No exemplo abaixo, o maior caminho possível para início na interseção 1 é $P = (1, 3, 5, 4, 7)$ e para início na interseção 5 é $P = (5, 3, 6)$ ou $P = (5, 4, 7)$.



Entrada

A primeira linha contém dois inteiros N e M, representando respectivamente o número de interseções e o número de ruas. As M linhas seguintes contém dois inteiros A e B indicando que existe uma ciclovia entre A a B.

Saída

Seu programa deve produzir uma única linha, contendo N inteiros $R_1, R_2, \ldots R_N$, onde R_i é o tamanho do maior caminho possível se o jogo começar na interseção i.

Restrições

- $1 \leq N \leq 10^5$.
- $0 \leq M \leq \dfrac{N(N-1)}{2}$.
- $0 \leq M \leq 5 \times 10^5$.
- $A \neq B$.
- $1 \leq A, B \leq N$.
- Não existem duas ciclovias iguais.

Informações sobre a pontuação

- Em um conjunto de casos de teste equivalente a 20 pontos, $N \leq 7$.
- Em um conjunto de casos de teste equivalente a 40 pontos, $N \leq 100$.
- Em um conjunto de casos de teste equivalente a 60 pontos, $N \leq 1000$.

Exemplos

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

Entrada	**Saída**
6 6 1 3 2 3 4 2 3 4 3 5 5 4	7 5 6 4 2 1

Entrada	**Saída**
7 6 1 2 1 3 3 5 3 6 5 4 4 7	5 6 4 2 3 2 2