

Universidade do Estado Santa Catarina (UDESC)

Universidade Federal do Paraná (UFPR)

Maratona Doméstica de Programação – 2009 – 1 (Dona Vanda)

<http://200.18.7.16:8001/boca>

Organizadores: Claudio (DCC), Rosso (DCC), André Guedes (DInf)

23 de maio de 2009



Lembrete:

- É permitido consultar livros, anotações ou qualquer outro material impresso cópia durante a prova.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa.
- Todos os problemas têm o mesmo valor na correção. Logo, leia e comece com os mais fáceis. Para esta prova, há vários fáceis.
- Procure resolver o problema de maneira eficiente. Na correção, a eficiência também é levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Utilize o *Clarification* para dúvidas da prova. Opcionalmente, os juízes respondem, contudo, os esclarecimentos são acessíveis a todos.
- Responsável pela construção da prova: Claudio – UDESC
- Revisão Técnica: André – UFPR

Agradecimentos

- Ao pessoal do suporte da UDESC, do DCC, Grupo Colméia, ...
- Pelas traduções: Fernando Deeke Sasse (DMAT), Alexandre (DCC), Adriano (DCC) e o Dênio (DCC).
- Ao patrocínio da TOTVS.
- Ao DCC e demais anônimos que tornaram este evento uma realidade.

Sumário

| | | |
|-----------|--|-----------|
| 1 | Problema A: Hide those Letters | 3 |
| 2 | Problema B: Ordenação <i>Mexida</i> | 4 |
| 3 | Problema C: Circunferência Através de Três Pontos | 5 |
| 4 | Problema D: Qualquer bobo pode fazê-lo | 7 |
| 5 | Problema E: Grupos Diedrais | 9 |
| 6 | Problema F: A Pizzaria do <i>Bocão</i> | 11 |
| 7 | Problema G: Deli–Deli | 12 |
| 8 | Problema H: Transparências do Professor <i>Claudius</i> | 13 |
| 9 | Problema I: Dominosa | 15 |
| 10 | Problema J: The Bottom of a Graph | 16 |

1 Problema A: Hide those Letters

¹
Arquivo: letters.[c|cpp|java]

Mrs Jones, a primary school teacher, has thought of a way to help her young pupils to learn to spell. She is removing certain letters from sentences and replacing them with underscore characters. The children have to write the appropriate letters in the spaces. As a starting point, Mrs Jones takes 2 different letters from a sentence (such as **a** and **e**), and the children have to work out whether each underscore represents an 'a' or an 'e', for example. Input to this problem will consist of a number of cases, terminated by a line containing just # #. Each case will begin with **two lower case letters** on a single line, separated by a space, followed by a line containing a single integer n ($1 \leq n \leq 100$) which gives the number of lines of text making up the case. The following n lines each contain a line of text of up to 255 characters. Output for each case will be the case number, followed by 1 line for each line of text presented in the input. The text will be as input but with every occurrence of either of the two letters replaced by an underscore character `_`. Although the letters provided are in lowercase, uppercase versions of the same letter must also be replaced. **Leave a blank line between cases.**

Sample Input

```
a e
2
Come here Evans, said the teacher.
I think not.
p r
1
Purple reindeer rarely appear in Ruritania.
# #
```

Sample Output

```
Case 1
Com_ h_r_ _v_ns, s_id th_ t__ch_r.
I think not.

Case 2
_u__le _eindee_ _a_ely a__ea_ in _u_itania.
```

¹Este problema é de fácil entedimento e resolução, assim, dispensamos a tradução. Basicamente, voce vai ajudar a sua professora primária a elaborar frases com letras faltando, previamente definidas. Assim, o estudante vai aprender a inserir letras certas, nas palavras que estão com espaços em branco.

2 Problema B: Ordenação *Mexida*

Arquivo: `mexidos_01.[c|cpp|java]`

Neste problema é dado uma série de listas contendo palavras e números. O objetivo é ordenar essas listas de tal modo a que todas as palavras fiquem em ordem alfabética e todos os números fiquem em ordem numérica. Além disso, se n -ésima posição na lista for um número ele deve continuar a ser um número, e se este for uma palavra deve continuar a ser uma palavra. Ou seja, é uma ordenação entre palavras e números, mas estes respeitam posições de palavras e números na lista.

Especificação da Entrada

A entrada irá conter várias listas, uma lista por linha. Cada elemento da lista será separado por uma vírgula seguida de um espaço, e a lista vai ser encerrada por um ponto. A entrada será encerrada por uma linha contendo apenas um único ponto.

Especificação da Saída

Para cada lista na entrada, a saída é uma lista ordenada *mexida*², separando cada elemento da lista, com uma vírgula seguida por um espaço, e finalizando a lista com um ponto.

Exemplo de Entrada

```
0.
banana, strawberry, OrAnGe.
Banana, StRaWbErRy, orange.
10, 8, 6, 4, 2, 0.
x, 30, -20, z, 1000, 1, Y.
50, 7, kitten, puppy, 2, orangutan, 52, -100, bird, worm, 7, beetle.
.
```

Exemplo de Saída

```
0.
banana, OrAnGe, strawberry.
Banana, orange, StRaWbErRy.
0, 2, 4, 6, 8, 10.
x, -20, 1, Y, 30, 1000, z.
-100, 2, beetle, bird, 7, kitten, 7, 50, orangutan, puppy, 52, worm.
```

²Como já escrito, onde tem palavras será uma palavra da lista, idem quanto a números.

3 Problema C: Circunferência Através de Três Pontos

Arquivo: `circle3.[c|cpp|java]`

Sua equipe deve escrever um programa que, dadas as coordenadas cartesianas de três pontos em um plano, encontre a equação da circunferência que passa por todos eles. Os três pontos não estarão sobre uma reta. Como solução, é impressa uma equação na forma:

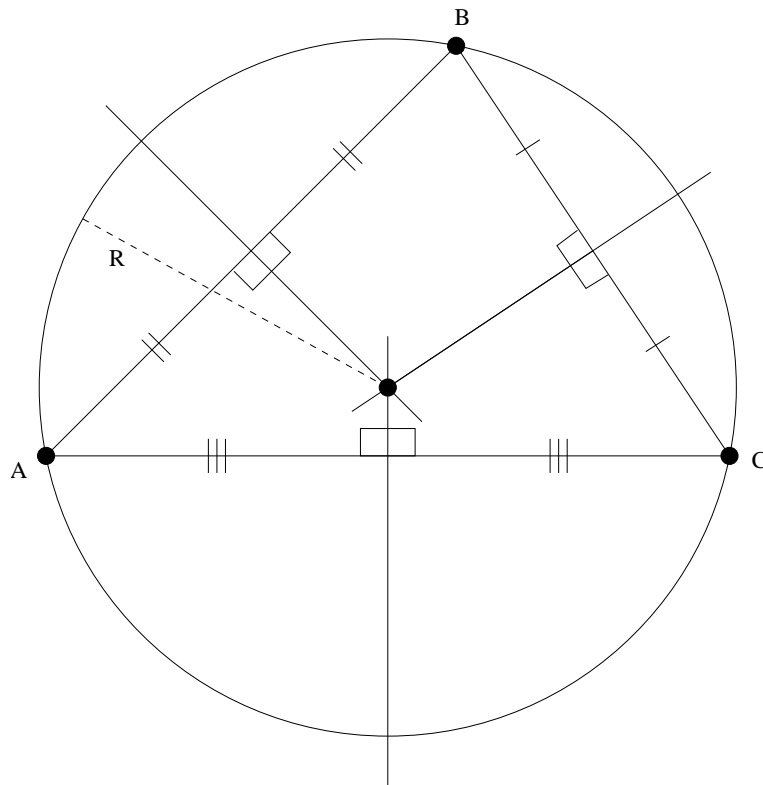
$$(x - h)^2 + (y - k)^2 = r^2 \quad (1)$$

e uma equação na forma

$$x^2 + y^2 + cx + dy - e = 0 \quad (2)$$

Cada linha de entrada para o seu programa conterá as coordenadas x e y de três pontos, na ordem $A_x, A_y, B_x, B_y, C_x, C_y$. Estas coordenadas são números reais separados entre si por um ou mais espaços.

Seu programa deve imprimir as equações requeridas em duas linhas usando o formato conforme o exemplo abaixo. Os valores calculados para h, k, r, c, d e e , nas Equações (1) e (2) acima, devem ser impressos com três dígitos depois da vírgula. Os sinais (mais, menos e igual) devem ser mudados conforme a necessidade para evitar múltiplos sinais antes de um número. Os sinais (mais, menos e igual) devem ser separados a partir dos caracteres vizinhos por um único espaço de cada lado. Outros espaços não devem aparecer nas equações. Imprima uma única linha em branco depois de cada equação.



Exemplo de entrada

```
7.0 -5.0 -1.0 1.0 0.0 -6.0
1.0 7.0 8.0 6.0 7.0 -2.0
```

Exemplo de saída

```
(x - 3.000)^2 + (y + 2.000)^2 = 5.000^2
x^2 + y^2 - 6.000x + 4.000y - 12.000 = 0
```

```
(x - 3.921)^2 + (y - 2.447)^2 = 5.409^2
x^2 + y^2 - 7.842x - 4.895y - 7.895 = 0
```

4 Problema D: Qualquer bobo pode fazê-lo

Arquivo: fool.[c|cpp|java]

Introdução

Certamente, você conhece alguém que pensa que é muito inteligente. Você pode desmascará-lo com o seguinte problema:

- *Você pode me dizer o que é um conjunto? Você pergunta para esta pessoa.*
- *Claro! Responde ele, um conjunto delimita uma lista de elementos (possivelmente vazia) através de abre e fecha chaves. Cada elemento pode ser ou um outro conjunto ou uma letra de um dado alfabeto. Elementos em uma lista são separados por vírgulas.*
- *Então, se eu te dar uma palavra, você pode me dizer se é uma representação sintaticamente correta de um conjunto?*
- *Claro, qualquer bobo pode fazê-lo!*

Agora você o pegou! Mostre para ele a seguinte gramática, que define sintaticamente um conjunto (que foi descrita informalmente por ele):

```
Set          ::= "{"ElementList" "  
ElementList ::= <empty> | List  
List         ::= Element | Element "," List  
Element      ::= Atom | Set  
Atom         ::= "{" | "}" | ","
```

Ou veja no formato *verbatim*:

```
Set          ::= "{" Elementlist " "  
Elementlist  ::= <empty> | List  
List         ::= Element | Element ", " List  
Element      ::= Atom | Set  
Atom         ::= "{" | "}" | ", "
```

<empty> significa a palavra vazia, ou seja, a lista no conjunto pode ser vazia. Logo ele percebe que a tarefa parece mais difícil que ele imaginava, pois percebe que o alfabeto consiste de caracteres que também são utilizados na definição sintática do conjunto. Assim, ele afirma que não é possível decidir de forma eficiente se uma palavra formada pelos símbolos "{", "}" e "," é ou não uma representação sintaticamente correta de um conjunto. Para discordar dele, você precisa escrever um programa eficiente que resolve este problema.

Especificação da Entrada

A primeira linha do arquivo de entrada contém um número representando o número de linhas do arquivo. Cada linha subsequente contém uma palavra, a qual o seu programa tem que dizer se essa palavra é uma representação sintaticamente correta de um conjunto. Assuma que cada palavra consiste de 1 a 200 caracteres do conjunto { "{", "}", ",", " " }.

Especificação da Saída

Apresente para cada caso de teste se a palavra é ou não um conjunto. A saída deve seguir o formato apresentado a seguir no exemplo.

Exemplo de Entrada

```
4
{}
{{{}}
{{{}} , { , }}
{ , , }
```

Exemplo de Saída

```
Word #1: Set
Word #2: Set
Word #3: Set
Word #4: No Set
```


5 Problema E: Grupos Diedrais

Arquivo: `diedros.[c|cpp|java]`

Considere n pontos em um círculo unitário com números rotulados por $k = 0, 1, \dots, n - 1$. Inicialmente o ponto k faz um ângulo de $360 \cdot k / n$ graus relativamente ao eixo x , medido no sentido anti-horário. São definidos dois tipos diferentes de operações sobre este conjunto de pontos.:

- rotação de $360/n$ graus na direção horária (**r**otation)
- reflexão com relação ao eixo x (**m**irror)

A figura 1 mostra um exemplo de tais operações:

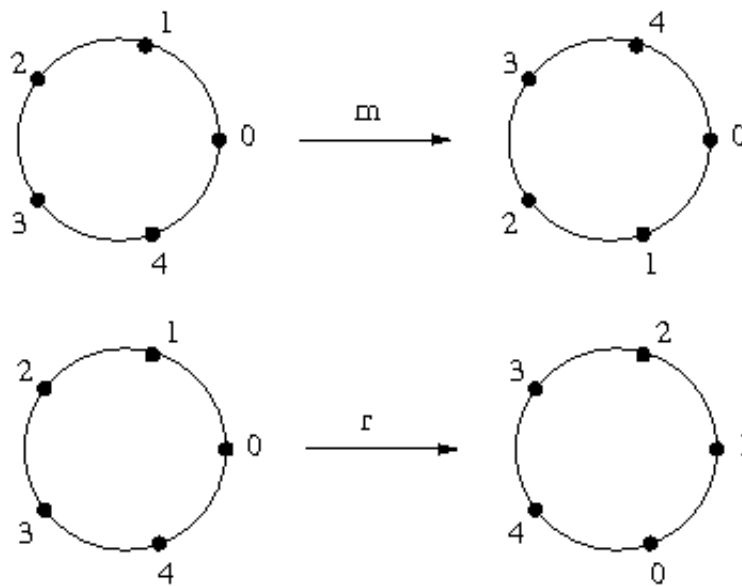


Figura 1: Reflexão (**m**) e rotação (**r**) de grupos diedrais

Dada uma sequência de operações, estamos interessados na menor sequência de operações que gera o mesmo resultado, ou seja, a posição de cada ponto é a mesma depois da execução de qualquer uma destas sequências de operações.

A sequência é dada como um string consistindo de caracteres 'r' e 'm' que representam rotações horárias e reflexões, respectivamente do inglês: *to the right* (**r**) and *mirror* (**m**). Ocorrências consecutivas múltiplas do mesmo caractere são condensadas na representação <caractere><número> e, por conveniência isso será também feito para ocorrências de um caractere. Assim, "rrmmrrrrrrrrrrrr" será abreviado por "r2 m1 r12". As representações de diferentes operações são sempre separadas por um espaço simples

Especificação da Entrada

O arquivo de entrada consiste de vários casos-testes. Cada caso-teste começa com uma linha contendo n ($3 \leq n \leq 10^8$), o número de pontos. A segunda linha de cada caso-teste consiste de uma sequência abreviada de operações como descrita anteriormente. Todos os números são positivos e menores que 10^8 . Não há linhas vazias no arquivo de entrada e nenhuma linha conterá mais do que 100000 caracteres. O último caso-teste é seguido por uma linha contendo 0.

Especificação da Saída

Para cada caso teste imprima uma linha contendo o formato abreviado da sequência com o número mínimo de operações que resulta na mesma configuração de pontos da entrada. No caso de múltiplas soluções ótimas, imprima qualquer solução.

Exemplo de Entrada

```
4
r2
100
m1 r100 m1
54
r218 m3 r1
0
```

Exemplo de Saída

```
r2

r1 m1
```

Sim, é uma linha em **branco** neste segundo teste, pois a sequência “m1 r100 m1” num círculo de 100 números, este volta a origem.

6 Problema F: A Pizzaria do *Bocão*

Arquivo: `bocao.[c|cpp|java]`

Tradicionalmente após a nossa competição doméstica, os juízes e os competidores vão há uma pizzaria favorita, Pizzaria do *Bocão*³. Os competidores estão realmente esfomeados após trabalharem arduamente durante cinco horas. Para conseguirem a sua pizza o mais rapidamente possível, eles então decidiram pedir a maior pizza para todos ao invés de várias pequenas. Contudo, eles se perguntam se é possível colocar esta grande pizza retangular sobre a superfície de uma mesa redonda de tal que esta não ultrapasse a borda da mesa. Escreva um programa para ajudá-los!

Especificação da Entrada

O arquivo de entrada contém vários casos teste. Cada caso de teste começa com um número inteiro r , o raio da superfície da mesa redonda na qual os competidores estão sentados. O término da entrada é dado por $r = 0$. Caso contrário, $1 \leq r \leq 1000$. Em seguida, tem 2 números inteiros l e w especificando a largura e comprimento da pizza, $1 \leq w \leq l \leq 1000$.

Especificação da Saída

Para cada caso de teste, voce deve indicar se a pizza pedida vai caber (*fits on the table.*) em cima da mesa ou não (*does not fit on the table.*). Siga o formato indicado no exemplo de saída. Uma pizza que apenas toca a borda da mesa sem interseção é considerada que esta cabe sobre a mesa, veja os 3 exemplos abaixo, que especificam o formato desejado.

Exemplo de Entrada

```
38 40 60
35 20 70
50 60 80
0
```

Exemplo de Saída

```
Pizza 1 fits on the table.
Pizza 2 does not fit on the table.
Pizza 3 fits on the table.
```

³Embora a boca seja redonda, todos sabem que a pizza de lá é retangular!

7 Problema G: Deli–Deli

Arquivo: `deli.[c|cpp|java]`

A Senhora Deli gerencia uma loja de doces chamada Deli–Deli. No último ano, a Senhora Deli decidiu expandir seu negócio e abriu uma loja de doces online. Ela contratou um programador que desenvolveu o web site para essa loja.

A pouco tempo, alguns de seus novos clientes online reclamaram a respeito das contas emitidas pelo sistema. O programador esqueceu-se de usar a forma plural para os casos em que o item era adquirido múltiplas vezes. Infelizmente o programador da Senhora Deli está de férias e agora é sua tarefa implementar essa nova feature para a Senhora Deli. A forma plural dá-se da seguinte maneira:

1. Se a palavra está na lista de palavras irregulares, substitua-a com o seu respectivo plural.
2. Senão, se a palavra termina em uma consoante seguida por “y”, substitua “y” por “ies”.
3. Senão, se a palavra termina em “o”, “s”, “ch”, “sh” ou “x”, acrescente “es” à palavra.
4. Senão, acrescente “s” à palavra.

Especificação da Entrada

A primeira linha do arquivo de entrada consiste de 2 inteiros L e N ($0 \leq L \leq 20$, $1 \leq N \leq 100$). As L seguintes linhas contêm a descrição das palavras irregulares e sua forma plural. Cada linha consiste de duas palavras separadas por um carácter espaço em branco, onde a primeira palavra é o singular, e a segunda palavra é a forma plural de alguma palavra irregular. Depois da lista de palavras irregulares, as N linhas seguintes contêm uma palavra cada, as quais você deverá flexionar em número (transformar para plural). Assuma que cada palavra consiste de no máximo 20 caracteres, em minúsculo, do alfabeto inglês (‘a’ to ‘z’).

Especificação da Saída

Imprima N linhas de saída, onde a i -ésima linha é a forma plural da i -ésima palavra de entrada.

| Exemplo de Entrada | Exemplo de Saída |
|---|---|
| 3 7 rice rice spaghetti spaghetti octopus octopi rice lobster spaghetti strawberry octopus peach turkey | rice lobsters spaghetti strawberries octopi peaches turkeys |

8 Problema H: Transparências do Professor *Claudius*

Arquivo: `slides.[c|cpp|java]`

O professor Claudius⁴ estará dando algumas aulas esta semana com suas transparências que juntou ao longo dos anos na academia. Ora feita por seus estudantes, ora por ele.

Infelizmente, ele não é uma pessoa muito *arrumada* (organizada) e colocou todas as suas transparências sobre uma grande pilha. Antes dele fazer a sua apresentação, ele tem que ordenar as lâminas/transparências. Contudo, como uma de suas características legais, é que ele é do tipo minimalista, ele quer fazer a classificação desse montante com o mínimo de trabalho possível.

A situação é a seguinte. Todas as lâminas (*slides*) tem números escritos sobre as mesmas de acordo com a sequência da fala do professor Claudius. Uma vez que as lâminas são colocadas umas sobre as outras e são transparentes, não se pode saber em qual transparência cada número está escrito.

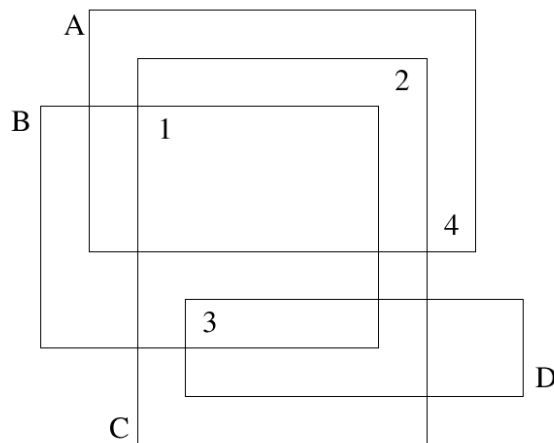


Figura 2: Transparências do professor Claudius

Bem, já que não se pode *ver* qual slide o número que está escrito, mas alguém pode deduzir quais números estão escritos em cada slide. Se nós etiquetarmos os slides pelas letras *A*, *B*, *C*, ... como na figura 2, é óbvio deduzirmos que a transparência *D* tem o número 3, *B* tem o número 1, a *C* tem o número 2 e *A* tem o número 4.

Sua tarefa, se você aceitar, é escrever um programa que automatize este processo.

Entrada

A entrada consiste de várias descrições de pilhas. Cada descrição de pilha começa com uma linha contendo um único inteiro n , o número de lâminas ou *slides* na pilha. Cada uma das n linhas seguintes contém quatro números inteiros x_{min} , x_{max} , y_{min} e y_{max} , valores que delimitam as coordenadas dos *slides*. Os *slides* serão rotulados/etiquetados como *A*, *B*, *C*, ... na ordem da entrada.

A sequência da entrada tem outras n linhas contendo dois inteiros cada, as coordenadas x e y de onde está impresso os n número da lâmina. A primeira coordenada para o número 1, a próxima para

⁴A brincadeira tem que ser comigo, se não entro no código de ética da UDESC.

o número 2, e assim sucessivamente. Nenhum número está escrito nos limites de um *slide*. Para um entendimento, confira a entrada 1 com o desenho da figura 2.

A entrada é terminada por uma pilha com a descrição começando com $n = 0$, a qual não deve ser processada.

Saída

Para cada descrição de pilha, escreva Heap (*pilha* em inglês) na entrada, um espaço em branco e seu número. Em seguida, imprima uma série de todos os *slides* cujos números podem ser unicamente determinados a partir da entrada. A ordem dos pares é feita pelas letras de seu identificador/rótulo.

Se nenhum casamento puder ser determinado a partir da entrada, basta imprimir a palavra none em sua linha. Escreva uma linha em branco na saída depois de cada teste.

Exemplo de Entrada

```
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
2
0 2 0 2
0 2 0 2
1 1
1 1
0
```

Exemplo de Saída

```
Heap 1
(A, 4) (B, 1) (C, 2) (D, 3)

Heap 2
none
```

9 Problema I: Dominosa

Arquivo: dominosa.[c|cpp|java]

A set of dominoes, that is, one instance of every unordered pair of the numbers from 0 to $n - 1$, has been arranged irregularly into a rectangle and the pattern has been recorded by writing the number in each square. For instance, in the figure below the original layout, using a ‘standard’ set ($n = 7$) of dominoes, is on the left and the puzzle configuration is on the right. Write a program to recreate the original configuration, guaranteed to exist and to be unique.



Input will consist of a series of puzzles. Each puzzle will start with a line containing an integer n ($2 \leq n \leq 12$). Following this will be n rows each containing $n + 1$ characters from the first $n + 1$ letters of the lower case alphabet, separated by single spaces. The set of puzzles will be terminated by a line containing a single zero (0). The above puzzle is represented in this form in the sample input.

Output will consist of a representation of the original grid in the same format as the input, except that a pair of letters constituting a horizontal domino are separated by an equals sign (=), as shown in the sample output below. Leave a blank line between successive grids.

| Sample input | Sample output |
|-----------------|-----------------|
| 7 | e=d g=g f d c=c |
| e d g g f d c c | b g=e a a a c f |
| b g e a a a c f | b b=a a a=g d g |
| b b a a a g d g | c e d b=c f d g |
| c e d b c f d g | a f d e e c b b |
| a f d e e c b b | d c=g a b e f=f |
| d c g a b e f f | g d=f b=f e e=c |
| g d f b f e e c | |
| 0 | |

10 Problema J: The Bottom of a Graph

Arquivo: `bottom.[c|cpp|java]`

We will use the following (standard) definitions from graph theory. Let V be a nonempty and finite set, its elements being called vertices (or nodes). Let E be a subset of the Cartesian product $V \times V$, its elements being called edges. Then $G = (V, E)$ is called a directed graph.

Let n be a positive integer, and let $p = (e_1, \dots, e_n)$ be a sequence of length n of edges $e_i \in E$ such that $e_i = (v_i, v_{i+1})$ for a sequence of vertices (v_1, \dots, v_{n+1}) . Then p is called a path from vertex v_1 to vertex v_{n+1} in G and we say that v_{n+1} is reachable from v_1 , writing $(v_1 \rightarrow v_{n+1})$.

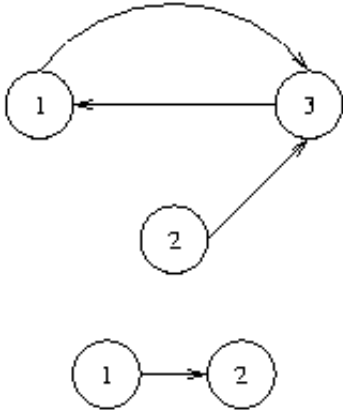
Here are some new definitions. A node v in a graph $G = (V, E)$ is called a *sink*, if for every node w in G that is reachable from v , v is also reachable from w . The bottom of a graph is the subset of all nodes that are sinks, i.e., $\text{bottom}(G) = \{v \in V \mid \forall w \in V : (v \rightarrow w) \Rightarrow (w \rightarrow v)\}$. You have to calculate the bottom of certain graphs.

Input Specification

The input contains several test cases, each of which corresponds to a directed graph G . Each test case starts with an integer number v , denoting the number of vertices of $G = (V, E)$, where the vertices will be identified by the integer numbers in the set $V = \{1, \dots, v\}$. You may assume that $1 \leq v \leq 5000$. That is followed by a non-negative integer e and, thereafter, e pairs of vertex identifiers $v_1, w_1, \dots, v_e, w_e$ with the meaning that $(v_i, w_i) \in E$. There are no edges other than specified by these pairs. The last test case is followed by a zero.

Output Specification

For each test case output the bottom of the specified graph on a single line. To this end, print the numbers of all nodes that are sinks in sorted order separated by a single space character. If the bottom is empty, print an empty line.

| | | |
|---|---|--|
|  | <p>Sample input</p> <pre> 3 3 1 3 2 3 3 1 2 1 1 2 0 </pre> | <p>Sample Output</p> <pre> 1 3 2 </pre> |
|---|---|--|