Pick up sticks – UVA 11686

Pega Varetas é um jogo fascinante. Uma coleção de varetas coloridas são despejadas sobre a mesa. Os jogadores se revezam tentando pegar uma vareta de cada vez sem mover qualquer uma das outras. É muito difícil de pegar uma se existe outra deitada no topo da mesma. Os jogadores, portanto, tentam pegar as varetas em uma ordem tal que eles nunca tem que pegar uma vareta debaixo de um outra.

Especificação de Entrada

A entrada consiste de vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros *n* e *m,* cada número pelo menos um e não superior a um milhão. O número inteiro *n* é o número de varetas, e *m* é o número de linhas que se seguem. As varas são numeradas de 1 a *n*. Cada uma das linhas seguintes contém um par de números inteiros *a*, *b*, indicando que há um ponto onde uma vareta *a* encontra-se em cima da vareta *b*. A última linha de entrada é 0 0. Estes zeros não são valores de *n* e *m*, e não deve ser processado como tal.

Exemplo de Entrada

```
3 2
1 2
2 3
0 0
```

Especificação de Saída

Para cada caso de teste, imprima *n* linhas de números inteiros, listando as varetas na ordem em que eles poderiam ser coletadas sem nunca pegar uma vareta com outra em cima dela. Se houver várias ordens corretas, qualquer uma servirá. Se não houver tal ordem correta, a saída de uma única linha contendo a palavra `IMPOSSIBLE`.

Exemplo de Saída

```
1
2
3
```

Ordering Tasks – UVA 10305

Input: standard input
Output: standard output
Time Limit: 1 second
Memory Limit: 32 MB

John tem *n* tarefas a fazer. Infelizmente, as tarefas não são independentes e a execução de uma tarefa somente é possível se outras tarefas já foram executadas.

Entrada

A entrada consistirá de várias instâncias do problema. Cada instância começa com uma linha contendo dois inteiros, **1 <= n <= 100** e **m**. **n** é o número de tarefas (enumeradas de **1** a **n**) e **m** é o número de relações de precedência direta entre as tarefas. Depois disto, haverá **m** linhas com dois inteiros **i** e **j**, representando o fato que a tarefa **i** deve ser executada antes da tarefa **j**. Uma instância com **n = m = 0** finalizará a entrada.

Saída

Para cada instância, imprima uma linha com **n** inteiros representando as tarefas em uma possível ordem de execução.

Exemplo de entrada

```
5 4
1 2
2 3
1 3
1 5
0 0
```

Exemplo de saída

```
1 4 2 5 3
```

Evaluate the Expression

Neste problema, iremos considerar uma expressão matemática de acordo com a seguinte gramática BNF:

```
<expression> = <variable> | <expression><operator><expression>| "("<expression>")"
<operator> = "+" | "*"
<variable> = "a" | "b" | "c" | … | "y" | "z"
```

Ao avaliar tais expressões, você seguirá regras convencionais. Isto significa que você terá que resolver primeiro as coisas que estão dentro de parêntesis e multiplicações antes de adições.

Exemplo: **`2*(3+4*2) = 22`**

Dada uma expressão e algumas inequações, você terá que atribuir a cada variável um valor inteiro positivo tal que o valor da expressão seja minimizado.

Considere como exemplo:

Expressão = `a+b*c e` inequações = `a>b`, `c>b`
Atribuição de: `a=2`, `b=1` e `c=2` nos dará o valor mínimo. **=> 2 + 1*2 = 4**

Input

A primeira linha de entrada é um inteiro **T**(**T**<100) que nos dá o número de casos de teste. Cada caso começa com uma linha que nos da a expressão. O comprimento da expressão é no máximo 300. A próxima linha contém um inteiro **I**(**I**<400) que informa o número de inequações. Cada uma das próximas **I** linhas informará uma inequação no formato *x*>*y* onde *x* e *y* são letras do alfabeto em minúsculo que estão presentes em uma dada expressão e *x* não é igual a *y*. Todas as inequações são distintas.

Output

Para cada caso, imprima o número do caso primeiro. Depois imprima o valor mínimo da expressão que pode ser obtida atribuindo um inteiro positivo para cada variável que segue as regras definidas pelas inequações. Você pode assumir que a saída caberá em um inteiro sinalizado de 32 bits. Se as inequações são inconsistentes, então imprima -1.

Exemplo de Entrada

```
3
a+b*c
2
a>b
c>b
z*(x+y)
3
z>x
x>y
z>y
a+b+c+a
0
```

Exemplo de Saída

```
Case 1: 4
Case 2: 9
Case 3: 4
```

Rare order – UVA 200

Um colecionador de livros raros descobriu recentemente um livro escrito em uma língua estranha que usava os mesmos caracteres do idioma Inglês. O livro continha um índice curto, mas a ordem dos itens no índice era diferente do que seria de se esperar se os caracteres fossem ordenados da mesma forma que no alfabeto Inglês. O colecionador tentou usar o índice para determinar a ordem dos caracteres (ou seja, a seqüência correspondente) do alfabeto estranho, então desistiu frustrado da tarefa tediosa.

Você terá que escrever um programa para completar o trabalho do colecionador. Em particular, o programa receberá um conjunto de *strings* classificadas de acordo com uma determinada sequência de agrupamento e você terá que determinar qual é esta seqüência.

Entrada

A entrada consiste de uma lista ordenada de *strings* em letras maiúsculas, uma por linha. Cada *string* contém no máximo 20 caracteres. O fim da lista é assinalado por uma linha com o caracter único '#'. Nem todas as letras são necessariamente usadas, mas a lista vai implicar uma ordem completa entre aquelas letras que são usadas.

Saída

A saída deverá ser uma única linha contendo letras maiúsculas na ordem em que especifica a sequência de agrupamento utilizado para produzir o arquivo de dados de entrada.

Exemplo de Entrada

```
XWY
ZX
ZXY
ZXW
YWWX
#
```

Exemplo de saída

```
XZYW
```