256 - Quadrados Peculiares – 93%

O número 3025 tem uma peculiaridade interessante: se você dividir sua representação decimal em duas strings de tamanho igual (30 e 25) e você elevar ao quadrado a soma desses números então você irá obter o número original:

$$(30 + 25)^2 = 3025$$

Este problema consiste em determinar todos os números com essa propriedade para um determinado número de dígitos.

Por exemplo, números com 4 dígitos variam de 0000 até 9999. Note que os zeros a esquerda devem ser considerados. Isto significa que 0001, que é igual a $(00 + 01)^2$, é um número peculiar com 4 dígitos. O número de dígitos será 2, 4, 6 ou 8. Note que o inteiro máximo (de 16 bits) é 32.767 e números de 8 dígitos serão processados. Um "bom" programador conseguiria fazer o processamento usando apenas inteiros. Além disso pense um pouco sobre a eficiência de seu programa.

Entrada

A entrada do seu programa será um arquivo texto contendo o número de dígitos (sendo 2, 4, 6 ou 8), cada número em uma linha.

Saída

A saída será composta por linhas contendo todos os números peculiares para cada número de dígitos de entrada. Os números peculiares deverão ser impressos em ordem crescente.
Atenção: é necessário imprimir os zeros a esquerda.

Exemplo de Entrada

```
2
2
6
```

Exemplo de saída

```
00
01
81
00
01
81
000000
000001
088209
494209
998001
```

344 - Numeração Romana – 92,8%

Muitas pessoas estão familiarizadas com algarismos romanos para números relativamente pequenos. Os símbolos “i”, “v”, “x”, “l”, e “c” representam os valores decimais 1, 5, 10, 50 e 100 respectivamente. Para representar outros valores, estes símbolos e múltiplos, quando necessários, são concatenados, com os símbolos mais significativos escritos da esquerda para a direita. Por exemplo, o número 3 é representado como “iii”, e o valor 73 é representado como “lxxiii”. A exceção à regra ocorre para número tendo valores unitários de 4 ou 9, e para os valores 40 ou 90. Para estes casos, a representação em números romanos é: “iv” (4), “ix” 9, “xl” (40) e “xc” (90). Então, a representação romana para 24, 39, 44, 49 e 94 é, respectivamente, “xxiv”, “xxxix”, “xliv”, “xlix” e “xciv”.

Os prefácios das páginas de livros são frequentemente numerados com números romanos, iniciando com “i” para a primeira página do prefácio, e continuando a sequência. Assuma que livros têm 100 ou menos páginas de prefácio. Quantos “i”, “v”, “x”, “l”, e “c” são necessários para numerar as páginas no prefácio? Por exemplo, num prefácio de cinco páginas nós usaremos os seguintes algarismos romanos: “i”, “ii”, “iii”, “iv” e “v”, significando que teremos que usar 7 caracteres “i”e 2 caracteres “v”.

Entrada

A entrada consistirá em uma sequência de inteiros de 1 até 100, terminando em 0. Para cada inteiro (exceto o zero final) determine o número de diferentes tipos de caracteres necessários para que os numerais romanos sejam impressos nos prefácios.

Saída

Para cada inteiro de entrada, escreva uma linha contendo o inteiro de entrada e o número de cada tipo de caractere necessário. Os exemplos a seguir ilustram a formatação necessária.

Exemplo de Entrada

```
1
2
20
99
0
```

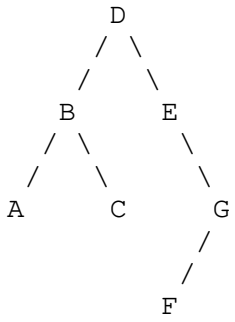
Exemplo de Saída

```
1: 1 i, 0 v, 0 x, 0 l, 0 c
2: 3 i, 0 v, 0 x, 0 l, 0 c
20: 28 i, 10 v, 14 x, 0 l, 0 c
99: 140 i, 50 v, 150 x, 50 l, 10 c
```

536 – Recuperação de Árvores – 94,5%

A pequena Valentina adora brinca com árvores binárias. Seu jogo favorito era construir de maneira aleatória árvores binárias com letras em cada nó.

Este é um exemplo de uma de suas criações:



Para armazenar suas árvores para as futuras gerações, ela escreveu duas strings para cada árvore: uma correspondendo ao percurso **em pré-ordem** e a outra string com o percurso **em ordem**.

Para a árvore ilustrada acima, o percurso em pré-ordem é **`DBACEGF`** e o percurso em ordem é **`ABCDEFG`**. Ela pensou que essa informação seria suficiente para reconstruir a árvore (porém ela nunca tentou fazer isso).

Agora, anos depois, olhando novamente para essas strings, ela percebeu que a reconstrução das árvores era possível, mas apenas porque ela nunca usou a mesma letra duas vezes em uma única árvore.

No entanto, fazer a reconstrução manualmente tornou-se rapidamente uma atividade tediosa.

Então, agora ela pede que você escreva um programa que faça este trabalho para ela!

Especificação da Entrada

A entrada consistirá em um ou mais casos de teste. Cada caso de teste consistirá em uma linha contendo duas strings presentando o percurso em pré-ordem e em ordem da árvore binária. As duas strings terão apenas letras maiúsculas e em cada uma não haverá repetição de letras (assim, cada string terá no máximo 26 caracteres).

A entrada termina com um fim-de-linha (*enter*).

Especificação da Saída

Para cada caso de teste, recupere a árvore binária da Valentina, representada pelas duas strings e imprima uma linha contendo a impressão em pós-ordem da árvore.

Exemplo de Entrada

```
DBACEGF ABCDEFG
BCAD CBAD
```

Exemplo de Saída

```
ACBFGED
CDAB
```

539 - Os Colonizadores de Catan – 94,9%

Dentro de *Os Colonizadores de Catan,* um jogo de tabuleiro alemão de 1995, jogadores deveriam tentar dominar uma ilha isolada construindo ruas, colonizadores e cidades através de lugares inexplorados.

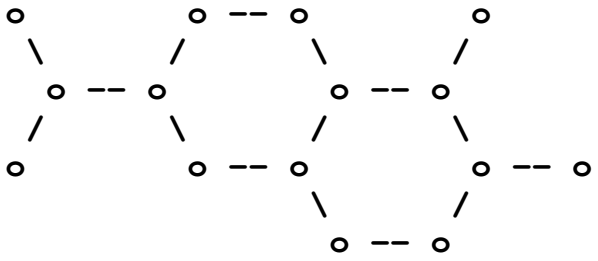
Você foi contratado por uma empresa de software que decidiu desenvolver uma versão para computador deste jogo, e você foi escolhido para implementar umas das regras desse jogo:

Quando o jogo termina, o jogador que tiver construído a estrada mais longa ganha dois pontos extras.

O problema aqui é que os jogadores tipicamente constroem redes complexas de estradas e não apenas caminhos lineares. Desta forma, determinar a estrada mais longa não é trivial (apesar de jogares humanos comumente identificarem essas estradas facilmente).

Comparado ao jogo original, nós resolveremos um problema simplificado aqui: você receberá um conjunto de nós (cidades) e o conjunto de arestas (segmentos de estrada) de tamanho fixo conectando os nós. A estrada mais longa é definida como o caminho mais longo dentro da rede que não use uma aresta duas vezes. Porém, nós podem ser visitados mais de uma vez.

Exemplo: a seguinte rede contém uma estrada de tamanho 12.



Entrada

O arquivo de testes conterá um ou mais casos de teste.

A primeira linha de cada caso de teste contém dois inteiros: o número de nós *n* (2 <= n <= 25) e o número de arestas *m* (1 <= m <= 25). As próximas *m* linhas descrevem as *m* arestas. Cada aresta é dada pelo número de dois nós que são conectados por ela. Nós são numerados de 0 até n-1. Arestas são não direcionadas. Nós têm grau menor ou igual a 3. A rede não é necessariamente conexa.

A entrada será encerrada com dois valores 0 (zero), para *n* e *m*.

Saída

Para cada caso de teste, imprima o tamanho da estrada mais longa (um tamanho para cada caso de teste, cada tamanho sozinho em uma linha).

Exemplo de Entrada

```
3 2
0 1
1 2
15 16
0 2
1 2
2 3
3 4
3 5
4 6
5 7
6 8
7 8
7 9
8 10
9 11
10 12
11 12
10 13
12 14
0 0
```

Exemplo de Saída

```
2
12
```

541 - Correção de Erro – 93,1%

Uma matriz booleana possui a *propriedade paridade* quando cada linha e cada coluna possuem uma soma par, isto é, contém um número par de bits com o valor *true*. A seguir, um exemplo de matriz 4 x 4 que possui essa propriedade:

```
1 0 1 0
0 0 0 0
1 1 1 1
0 1 0 1
```

As somas das linhas são 2, 0, 4 e 2. A somas das colunas são 2, 2, 2 e 2.

Seu trabalho é escrever um programa que leia uma matriz e verifique se ela possui paridade. Se não, seu programa deverá verificar se a propriedade de paridade pode ser estabelecida mudando apenas um bit. Se isso não for possível, a matriz deve ser classificada como corrompida.

Entrada

O arquivo de entrada irá conter um ou mais casos de teste. A primeira linha de cada caso de teste conterá um inteiro *n* (*n*<*100*), representando o tamanho da matriz. Nas próximas *n* linhas, haverá *n* inteiros por linha, com valores 0 ou 1. A entrada se encerrará quando, no lugar do valor *n* for recebido o valor 0.

Saída

Para cada matriz de entrada, imprima uma linha. Se a matriz já possuir a propriedade paridade, imprima "*OK*". Se a propriedade paridade puder ser estabelecida com a mudança de um bit, imprima "*Change bit (i,j)*" onde *i* é a linha e *j* a coluna do bit que precisa ser mudado. Caso contrário, imprima "*Corrupt*".

Exemplo de Saída

```
4
1 0 1 0
0 0 0 0
1 1 1 1
0 1 0 1
4
1 0 1 0
0 0 1 0
1 1 1 1
0 1 0 1
4
1 0 1 0
0 1 1 0
1 1 1 1
0 1 0 1
0
```

Exemplo de Saída

```
OK
Change bit (2,3)
Corrupt
```

558 - Buracos de Minhoca – 93%

No ano 2163, buracos de minhoca (*wormholes*) foram descobertos. Um wormhole é um túnel subespacial através do espaço-tempo conectando dois sistemas estelares. Os wormholes têm algumas propriedades peculiares:

- Wormholes são "mão única".
- O tempo necessário para atravessar um wormhole é desprezível.
- Um wormhole tem duas extremidades, cada uma situada em um sistema estelar.
- Um sistema estelar pode ser a extremidade de mais de um wormhole.
- Por algum motivo desconhecido, partindo-se do nosso sistema solar é sempre possível atingir a qualquer outro sistema seguindo uma sequência de wormholes (talvez a Terra seja o centro do universo).
- Entre qualquer par de sistemas estrelares, há no máximo um wormhole em cada direção.
- Nenhum wormhole tem suas duas extremidades no mesmo sistema solar.

Todos os wormholes têm uma diferença de tempo constante entre suas extremidades. Por exemplo, um wormhole específico faz com que a pessoa viajando através dele viaje 15 anos no futuro. Outro wormhole pode fazer com que a pessoa viaje 42 anos no passado.

Um físico brilhante, que vive na Terra, deseja usar os wormholes para estudar o Big Bang. Já que a dobra espacial ainda não foi inventada, não é possível para ele viajar de um sistema estelar para outro diretamente. Mas isto pode ser feito usando wormholes, é claro.

O cientista deseja encontrar um ciclo de wormholes de forma que ele consiga ser levado para o passado. Viajando através desse ciclo várias vezes, o cientista poderá voltar o quanto quiser no tempo até chegar no início do universo e ver o Big Bang com seus próprios olhos. Escreva um programa que identifique se esse ciclo existe.

Entrada

A entrada iniciará com uma linha contendo a quantidade de casos de entrada ***c*** a serem analisados. Cada caso inicia com uma linha com dois números ***n*** e ***m***. Estes valores indicam o número de sistemas estelares ($1 \leq n \leq 1000$) e o número de wormholes ($0 \leq m \leq 2000$). Os sistemas estelares são numerados de 0 a n-1. Para cada wormhole uma linha contendo três números inteiros, **x**, **y** e **t** será dada. Estes números indicam que este wormhole permite que alguém viaje do sistema estelar x para o sistema estelar y saindo t anos no futuro ou passado ($-1000 \leq t \leq 1000$).

Saída

A saída consistirá em **c** linhas, sendo uma linha para cada caso de teste, contendo a palavra `possible` se for possível voltar no tempo indefinidamente, ou `not possible` se não for possível fazer isso considerando o conjunto de sistemas estelares e de wormholes dados.

Exemplo de Entrada

```
2
3 3
0 1 1000
1 2 15
2 1 -42
4 4
0 1 10
1 2 20
2 3 30
3 0 -60
```

Exemplo de Saída

```
possible
not possible
```