# ToPAS'15
# Torneio de Programação para Alunos do Secundário

## Departamento de Ciência de Computadores

http://www.dcc.fc.up.pt/topas/

## Conjunto de Problemas


Faculdade de Ciências da Universidade do Porto

8 de maio de 2015

Este conjunto de problemas deverá conter sete (7) problemas e dezasseis (16) páginas.
Se faltar algum problema, por favor avise a organização.

Edição realizada em colaboração com o Departamento de Engenharia Electrónica e Informática, Faculdade de Ciências e Tecnologia, Universidade do Algarve.

# ToPAS'15
# Torneio de Programação para Alunos do Secundário

Dep. Ciência de Computadores – FCUP
8 de maio de 2015

## Conteúdo

**Problema A**: Doentes da bola . . . . . 3
**Problema B**: Varas e côvados . . . . . 5
**Problema C**: Preço incerto . . . . . 7
**Problema D**: O estranho caso dos símbolos mutantes . . . . . 9
**Problema E**: Leilões Morse . . . . . 11
**Problema F**: Batalha Naval . . . . . 13
**Problema G**: *iNetCode this message* . . . . . 15

# Doentes da bola

Com o aproximar do fim do campeonato os doentes da bola agarram-se à calculadora para saberem se já podem festejar ou a trabalhar na racionalização. As contas nem são difíceis de fazer. Em cada uma das jornadas em falta estão 3 pontos em disputa. Se o adversário mais próximo obtiver todos esses pontos, a nossa equipa nenhum, e mesmo assim tivermos mais pontos, então o campeonato já é nosso. Se for ao contrário, temos dizer adeus ao campeonato. Mas também pode acontecer que se perdermos pontos e eles ganharem qualquer das equipas ainda possa ganhar.

## Tarefa

Dados os pontos da nossa equipa, do adversário mais próximo, e o número de jornadas em falta (havendo 3 pontos em disputa em cada uma delas), determinar em qual dos casos está a nossa equipa, identificado por um *smiley*, a saber:

`:-D` somos campeões – mesmo que percamos todos os jogos e eles os ganhem todos, não conseguirão ter mais pontos do que nós;
`:-)` somos os maiores – temos mais pontos que o adversário e eles só vencem se nós perdermos demasiados pontos;
`:-|` ainda há esperança – temos menos pontos que o adversário mas ainda podemos ser vencedores;
`:-(` para o ano é a nossa vez – mesmo que ganhemos todos os jogos e eles percam todos já não temos hipótese.

Para simplificar as contas vamos supor que a equipa neste momento com mais pontos tem um número de golos muito maior (imbatível). Sendo assim, se o número de pontos no final do campeonato for igual, será campeã a equipa que neste momento tem mais pontos. Vamos também supor que neste momento as equipas têm pontuações diferentes.

## Input

Há apenas uma linha de input com 3 inteiros, nesta ordem:

- o número de pontos da nossa equipa
- o número de pontos do nosso adversário mais próximo
- o número de jornadas a disputar

## Output

Apenas uma linha de output com um *smiley* apropriado, consoante a diferença pontual e a explicação acima.

### Exemplo 1

**Input**

```
30 20 3
```

**Output**

```
:-D
```

### Exemplo 2

**Input**

```
30 21 3
```

**Output**

```
:-D
```

### Exemplo 3

**Input**

```
30 22 3
```

**Output**

```
:-)
```

### Exemplo 4

**Input**

```
22 30 3
```

**Output**

```
:-|
```

### Exemplo 5

**Input**

```
21 30 3
```

**Output**

```
:-(
```

### Exemplo 6

**Input**

```
20 30 3
```

**Output**

```
:-(
```

# Varas e côvados

O sistema métrico foi adotado em Portugal há pouco mais de cento e sessenta anos, mais exatamente em 1852, durante o reinado de D. Maria II. Antes do metro, as unidades de comprimento usadas em Portugal vinham da idade média e eram semelhantes às ainda hoje comuns em alguns países anglo-saxónicos.

O anterior sistema, antes de D. Maria II, baseava-se no *palmo*, que correspondia mais ou menos à distância entre a ponta do polegar e a ponta do dedo mínimo numa mão aberta. Além do palmo, havia o *côvado*, que são 3 palmos, e a *vara*, que são 5 palmos. Acima da vara, estava a *braça*, que são duas varas. Por outro lado, abaixo do palmo havia a *polegada*: 8 polegadas fazem um palmo. De novo acima do palmo, tínhamos o *pé*, que são 12 polegadas, o *passo*, que são 5 pés, e a *toesa*, que são 6 pés.

Portanto, dispúnhamos de oito unidades e nem todas eram múltiplas ou submúltiplas umas das outras.

Para nós, tão habituados aos metros, quilómetros, centímetros, o sistema medieval parece muito complicado, mas com certeza que os nossos trisavós não se atrapalhavam.

**Nota:** Na foto, observamos a medida de uma vara e a medida de um côvado, gravadas na pedra, em Sortelha.

## Tarefa

Escreva um programa que, dada uma medida de comprimento expressa em palmos, côvados, varas, braças, polegadas, pés, passos e toesas, calcule a expressão dessa medida apenas usando varas, palmos e polegadas. No resultado, o número de polegadas deve ser menor que número de polegadas num palmo e o número de palmos deve ser menor que o número de palmos numa vara.

## Input

Uma linha contendo oito número inteiros não negativos, representando a medida de comprimento que queremos analisar: primeiro é o número de braças, o segundo é o número de toesas, o terceiro é o número de passos, o quarto é o número de varas, o quinto é o número de côvados, o sexto é o número de pés, o sétimo é o número de palmos e o oitavo é o número de polegadas.

## Output

Uma linha que contém três números não negativos, representado a medida de comprimento usando apenas varas, palmos e polegadas: o primeiro número é o número de varas, o segundo é o número de palmos e o terceiro é o número de polegadas.

## Exemplo

### Input

```
0 4 0 0 0 7 0 3
```

### Output

```
9 1 7
```

# Preço incerto


O João e os seus amigos estão a criar uma app para ajudar a poupar no custo dos combustíveis automóveis. Neste momento a app já consegue obter o preço por litro dos combustíveis em todas as bombas localizadas num certo raio, bem como a distância a que estas se encontram. O problema é que muitas vezes a bomba com o melhor preço fica tão longe que a poupança se dissiparia no custo da deslocação.

A tarefa do João é determinar a bomba de gasolina a recomendar ao condutor que utiliza a app, sabendo que quantidade quer abastecer e qual o consumo médio do automóvel. Por simplicidade, assume-se que após abastecer, o condutor volta ao ponto partida.

## Tarefa

Dada a quantidade de gasolina que pretende abastecer e o consumo do automóvel, e ainda uma lista de postos de abastecimento com indicação da sua distância ao ponto atual e do preço por litro do combustível, determinar a bomba de gasolina que permite um menor custo por litro efetivo (em cêntimos, com arredondamento por defeito).

O *custo por litro efetivo* contabiliza também o custo da deslocação até à bomba de gasolina (ida e volta). Assim, se pretender abastecer 50 litros mas tiver de fazer 5 quilómetros até à bomba, e outros 5 para voltar, e o automóvel gastar 10 litros de gasolina por cada 100 quilómetros (isto é, 1 litro por cada 10 Km) então, no final, era como se tivesse ficado apenas com 49 litros dos 50 que abasteceu. Aos 49,5 litros efetivamente disponíveis no fim, há que acrescentar o meio litro que gastou na ida. Se o preço por litro cobrado na bomba for de 1,30 euros então pagaria 65 euros pelos 50 litros, pelo que o preço por litro efetivo era de 132 cêntimos, o que corresponderia a pagar 65 euros pelos 49 litros. Na estimativa, assume-se que o preço da gasolina que já tem no depósito antes de abastecer é igual ao que iria pagar. O custo por litro efetivo é obtido por arredondamento para o maior inteiro não superior (por exemplo, $132,7$ e $132,1$ cêntimos seriam ambos arredondados para 132 cêntimos).

## Input

A linha inicial contém dois inteiros: o primeiro é a quantidade a abastecer (em litros) e o segundo o consumo do automóvel (litros por cada 100 Km). A segunda linha contém um inteiro $n$ que indica o número de bombas de gasolina a considerar, sendo $1 < n < 2000$. Cada uma das $n$ linhas seguintes, tem dois inteiros, referentes a uma bomba: a distância a que se encontra (em metros) e o preço por litro de combustível (em cêntimos). Supõe-se que esse combustível é o que o condutor pretende usar.

**Nota:** Os casos de teste não serão sensíveis a perdas de precisão, podendo realizar os cálculos intermédios em vírgula flutuante.

### Output

O número da bomba a recomendar, na ordem em que estas foram listadas nos dados, sendo a bomba 1 a primeira a ser listada. Se houver mais do que uma bomba com o *melhor preço efetivo*, será apresentada a que se encontrar mais perto. Se mesmo assim existir mais do que uma nessas condições, indicará a que estiver primeiro na lista.

### Exemplo 1

**Input**

```
50 6
7
8231 135
9542 132
4143 139
5700 134
6532 140
7225 139
5700 134
```

**Output**

```
4
```

### Exemplo 2

**Input**

```
45 5
3
7500 137
9400 135
900 138
```

**Output**

```
2
```

### Exemplo 3

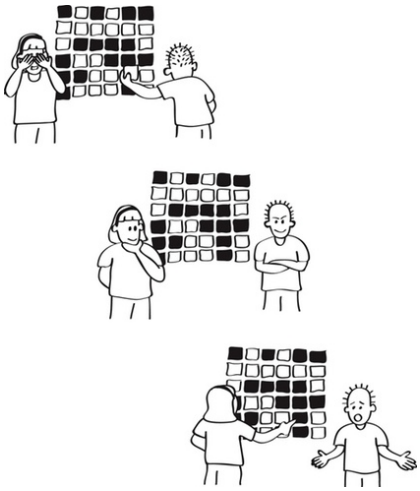
**Input**

```
40 5
3
7500 137
9400 135
900 138
```

**Output**

```
3
```

# O estranho caso dos símbolos mutantes



A Alice inventou um código constituído por sequências de símbolos (zeros e uns), que utiliza para a troca de mensagens secretas com o Bernardo. A Teresa Coscuvilheira, que gosta de estar a par de tudo que acontece e que não acontece, ficou muito frustada quando se viu incapaz de decifrar as mensagens que a Alice deixava na secretária do Bernardo. Como vingança por tamanha desfaçatez, decidiu adulterar as mensagens secretas da Alice. Sempre que encontrava um novo bilhete na secretária, apressava-se a trocar `1`'s por `0`'s e `0`'s por `1`'s, mas não muitos para o Bernardo e a Alice não suspeitarem.

Apercebendo-se deste estranho fenômeno de símbolos mutantes — poucos, mas suficientes para retirar o sentido às mensagens —, a Alice arranjou uma solução para permitir ao Bernardo corrigir, ou pelo menos, detetar alguns casos de mutações. A solução é a seguinte. Por exemplo, para uma mensagem $s_1\, s_2\, s_3\, s_4\, s_5\, s_6\, s_7\, s_8\, s_9$, constituída por uma sequência de nove símbolos (`0`'s e `1`'s), a Alice passou a acrescentar sete novos símbolos $p_1\, p_2\, p_3\, p_4\, p_5\, p_6\, p_7$. O cálculo dos novos símbolos é feito assim. Os nove símbolos originais são agrupados numa matriz quadrada (de dimensão 3x3, neste exemplo). A cada uma das linhas é acrescentado um novo símbolo $p_i$, com valor `0` ou `1`, por forma a que o número total de `1`'s numa linha seja par. Para as colunas a estratégia é a mesma. A cada uma das colunas é acrescentado um novo símbolo $p_i$ com valor `0` ou `1`, por forma a que o número total de `1`'s em cada coluna seja par.

$$\begin{array}{c|c} s_1\, s_2\, s_3 & p_4 \\ s_4\, s_5\, s_6 & p_5 \\ s_7\, s_8\, s_9 & p_6 \\ \hline p_1\, p_2\, p_3 & p_7 \end{array}$$

A mensagem a transmitir seria então constituída por 16 símbolos agrupados da seguinte forma $s_1\, s_2\, s_3\, s_4\, s_5\, s_6\, s_7\, s_8\, s_9\, p_1\, p_2\, p_3\, p_4\, p_5\, p_6\, p_7$, ou seja, primeiro tem a sequência original e depois a de controlo. Se na análise da mensagem for detetado um erro de paridade numa só linha e numa só coluna, então o erro fica localizado.

## Tarefa

Dada uma mensagem recebida, codificada conforme se descreveu acima, indicar:

- se não foram detetados erros;
- ou se foi detetado um erro, indicando a posição onde ocorreu na mensagem recebida;
- ou se foi detetado mais do que um erro.

### Input

Uma linha constituída por um número inteiro $n$, um espaço, e uma sequência de $(n+1)^2$ caracteres (`0`'s e `1`'s) correspondentes à mensagem recebida, sendo $2 \leq n \leq 50$. Note que $(n+1)^2 = n^2 + 2n + 1$ e a matriz quadrada formada com a mensagem recebida tem $n+1$ linhas e colunas.

### Output

Uma única linha com uma das mensagens seguintes (sem aspas):

| | |
|---|---|
| "`OK`" | – se não foram detetados erros. |
| "`ERRO EM` $k$" | – se foi detetado um erro na posição $k$, com $1 \leq k \leq (n+1)^2$. $k$ é a posição na mensagem recebida onde foi detetado (e corrigido) o erro. |
| "`ERROS`" | – caso tenha sido detetado mais do que um erro. |

### Exemplo 1

**Input**

```
2 100111110
```

**Output**

```
OK
```

### Exemplo 2

**Input**

```
2 000111110
```

**Output**

```
ERRO EM 1
```

### Exemplo 3

**Input**

```
2 000111111
```

**Output**

```
ERROS
```

### Exemplo 4

**Input**

```
3 1001011111101010
```

**Output**

```
OK
```

### Exemplo 5

**Input**

```
3 1011011111101010
```

**Output**

```
ERRO EM 3
```

### Exemplo 6

**Input**

```
3 1010011111101010
```

**Output**

```
ERROS
```

### Exemplo 7

**Input**

```
2 100111100
```

**Output**

```
ERRO EM 8
```

### Exemplo 8

**Input**

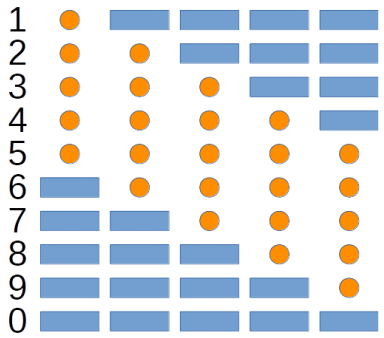
```
5 00101101001011010101001000111100111
```

**Output**

```
ERROS
```

# Leilões Morse

A leiloeira TopasLeilões inaugurou recentemente um sistema eletrónico para controlar os leilões que organiza. Com o novo sistema, os compradores fazem os seus lances usando um aparelho electrónico muito simples, onde existe apenas um botão. Carregando brevemente no botão, o aparelho emite um "bip" eletrónico curto, que representa o carácter ponto, '.'; carregando mais durante mais tempo, o aparelho emite um "bip" mais longo, que representa o carácter traço, '-'.




Usando sequências de pontos e traços, os compradores comunicam ao sistema o valor dos seus lances, usando o código Morse.

## Tarefa

Escreva um programa que aceite da consola a sequência de lances recebidos pelo sistema e escreva, também na consola, após cada lance, o valor do lance mais alto recebido até à altura e o número de lances válidos. Um lance é válido se a sequência de pontos e traços representar inequivocamente um número, de acordo com a tabela de códigos Morse dos algarismos decimais:

| | |
|---|---|
| 1 | ·---- |
| 2 | ··--- |
| 3 | ···-- |
| 4 | ····- |
| 5 | ····· |
| 6 | -···· |
| 7 | --··· |
| 8 | ---·· |
| 9 | ----· |
| 0 | ----- |

## Input

O input é uma sequência de linhas contendo apenas carateres '.' (ponto) e '-' (hífen, traço, sinal de menos). Cada linha representa um lance, que pode ser válido ou inválido, tal com explicado na secção precedente. A sequência termina quando chega uma linha com cinco traços, representando o número zero. Um ficheiro tem no máximo 100 linhas e cada linha tem no máximo 40 caracteres.

## Output

O output é uma sequência de linhas, uma linha por cada linha do input, excluindo a linha de terminação. Em cada linha vêm dois números separados por um espaço: o primeiro representa

o maior lance até à linha de input correspondente e o segundo o número de lances válidos, até essa linha de input. Se não houver lances válidos até essa linha, o valor do maior lance válido será 0 até essa linha.

### Exemplo 1

**Input**

```
.---------
-----
```

**Output**

```
10 1
```

### Exemplo 2

**Input**

```
.---------
.-.-......
-----
```

**Output**

```
10 1
10 1
```

### Exemplo 3

**Input**

```
.---------
.-.-......
..........
-----
```

**Output**

```
10 1
10 1
55 2
```

### Exemplo 4

**Input**

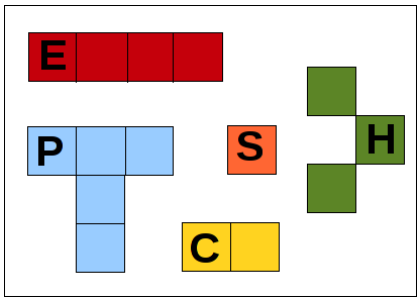
```
.---.--
.----.-
-----
```

**Output**

```
0 0
0 0
```

# Batalha Naval

Não havia Game Boy, PlayStation, New Nintendo 3DS, Wii, PADs, Smartphones, nada, nada ... papel e lápis bastava para dar largas à imaginação e assegurar umas horas de entretenimento. Não falamos do período Jurássico. Segundo alguns registos, o jogo Batalha Naval surge na Primeira Guerra Mundial (1914–1918). É, assim um jogo de má memória mas, contudo, centenário.




Na Batalha Naval participam dois jogadores, cada um com a sua frota e o objetivo é afundar a frota do oponente, adivinhando a posição dos navios inimigos.

Cada jogador dispõe a sua frota numa grelha e mantém também uma outra grelha que representa a disposição da frota do adversário (segundo o que for descobrindo dela durante o jogo). No posicionamento da frota, pode optar por qualquer orientação para cada navio, na vertical ou horizontal, desde que os navios não se toquem em nenhum ponto.

Existem várias versões do jogo, que diferem na dimensão da grelha, na constituição da frota ou nas regras do jogo. Vamos assumir que a grelha é $10 \times 10$. Em cada jogo, a constituição da frota é igual para ambos os jogadores, mas admitimos alguma flexibilidade no tipo e número de navios a usar. Os tipos possíveis são: porta-aviões (**P**), encouraçados (**E**), hidroaviões (**H**), cruzadores (**C**) e submarinos (**S**). A figura mostra as suas formas. Os jogadores decidirão quantos usam de cada tipo. Quanto às regras, assumimos que, cada jogador dá três tiros em cada jogada e depois passa a vez ao adversário, a menos que a sua jogada seja inválida. Em resposta, o adversário indica-lhe se os três tiros acertaram na água ou o número de tiros que atingiram cada tipo de navio atingido (sem dizer quantos navios atingiu de cada tipo).

## Tarefa

Como papel-e-lápis e jogos presenciais parecem estar fora de moda, vamos criar um programa simples para dar suporte à Batalha Naval *online*. O programa recebe a disposição da frota de cada jogador e uma sequência de jogadas e dá o *feedback* que o adversário daria a cada jogada. A sequência termina se um dos jogadores vencer o jogo mas pode também terminar em qualquer fase do jogo. O programa tem de identificar a frota escolhida.

## Input

Começa por 10 linhas, com 20 inteiros separados por espaços, que são `0` (água) ou `1` (navio). A parte esquerda define o posicionamento da frota do jogador 1 e a direita define o posicionamento da frota do jogador 2. A seguir tem uma linha com um inteiro positivo $m$ que representa o número total de jogadas. Depois, tem $m$ linhas, cada uma com três pares de inteiros positivos. Cada par corresponde a um tiro e identifica a linha e coluna da posição a atingir (as linhas e colunas são numeradas a partir de 1, de cima para baixo e da esquerda para a direita). Começa sempre a jogar o *jogador 1*. Se algum dos tiros de uma jogada referir uma posição já anteriormente atingida pelo jogador (navio ou água), essa jogada é considerada totalmente inválida e, na linha seguinte, estará uma jogada do mesmo jogador.

### Output

Para cada jogada, tem uma linha com o formato "`Jogador k: Agua`", "`Jogador k: Invalida`", "`Jogador k: seq`", onde `k` será substituído pelo identificador do jogador e `seq` pela informação sobre o número de tiros que atingiu cada tipo de navio atingido. As aspas não são incluídas nem acentos. A sequência `seq` é uma sequência de pares "$Tt$", com um espaço entre cada par mas não entre $T$ e $t$, sendo $T$ o identificador de um tipo de navio e $t$ o número de tiros que atingiram navios desse tipo nessa jogada ($t = 1, 2, 3$). A sequência deverá estar ordenada, por tipo de navio (segundo a ordem `P`, `E`, `H`, `C`, `S`).

Se algum dos jogadores vencer o jogo, por destruição da frota do adversário, o output terá mais uma linha com "`Venceu o jogador k`", sendo `k` o identificador do vencedor.

## Exemplo 1

### Input

```
1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0
1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0
6
9 1 9 8 5 3
1 1 1 2 1 3
8 2 9 3 9 9
1 4 7 8 1 10
10 2 9 7 8 8
9 1 9 3 8 1
```

### Output

```
Jogador 1: H2 C1
Jogador 2: E3
Jogador 1: H1
Jogador 2: P1 E1
Jogador 1: H1
Jogador 2: S1
```

## Exemplo 2

### Input

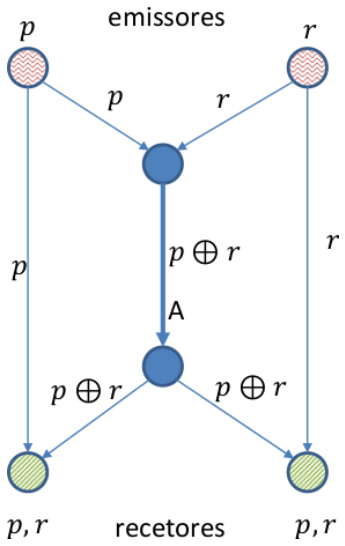
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
11
1 5 3 5 6 9
1 4 4 4 10 3
8 8 8 6 3 4
2 4 4 4 4 3
2 4 6 4 6 5
1 5 10 7 9 7
5 5 10 7 7 9
4 6 5 6 6 6
8 7 9 6 7 6
6 3 4 5 1 1
3 3 2 4 4 4
```

### Output

```
Jogador 1: Agua
Jogador 2: Agua
Jogador 1: P2 C1
Jogador 2: Invalida
Jogador 2: C1
Jogador 1: Invalida
Jogador 1: Agua
Jogador 2: P2
Jogador 1: P3
Jogador 2: Agua
Jogador 1: C1
Venceu o jogador 1
```

# *iNetCode this message*

As redes de comunicação estão cada vez mais ocupadas e as aplicações requerem um crescendo de largura de banda. Com o objetivo de otimizar o uso das redes e conseguir transportar o maior número de bytes possíveis nas mesmas ligações, desenvolveu-se a codificação de rede (*network coding*).




Na codificação de rede não se envia os dados exatamente como são gerados pelas aplicações, mas envia-se sim combinações de pedaços desses dados. Desde que as combinações de pedaços recebidas sejam independentes, consegue-se recuperar os dados originais. Um exemplo de operador para essas combinações (um "combinador") é o XOR ($\oplus$), que corresponde ao *ou-exclusivo*.

O interessante do XOR é a sua propriedade lógica de que que $p \oplus p = 0$. Ou seja, quando aplicamos a operação ao mesmo elemento (valor) obtemos zero. E quando operamos com zero obtemos o valor dado, i.e., $p \oplus 0 = p$. Por isso, se tivermos $p$ e $r$ temos:

$$\left.\begin{array}{ll} p \oplus p & = 0 \\ p \oplus 0 & = p \end{array}\right\} \Rightarrow \quad p \oplus r \oplus p = r \oplus (p \oplus p) = r$$

Na figura, podemos ver como se consegue alguma poupança nas mensagens enviadas. Os nós emissores têm, cada um, dados diferentes ($p$ e $r$) para enviar para ambos os nós recetores. Na ligação A, usando a codificação de rede, apenas precisamos de enviar uma mensagem com $p \oplus r$ em vez de enviarmos duas mensagens separadas $p$ e $r$. E, os nós recetores conseguem descodificar o $p$ e o $r$ usando as propriedades indicadas.

## Tarefa

Este problema pede que ponham as vossas mentes matemáticas a trabalhar para desenvolver um programa que dada uma lista de mensagens recebidas, descodifique o máximo possível de dados individuais. Serão dadas as combinações de dados nas várias mensagens recebidas. Como saída devem ser apresentados os dados individuais que se conseguiram descodificar (por ordem alfabética). Cada letra (sempre minúscula) representa um dado diferente. As várias mensagens são fornecidas em linhas separadas, e cada mensagem pode conter a combinação de várias letras.

## Input

Na primeira linha tem um inteiro $n$ que é o número de mensagens recebidas, com $2 \leq n \leq 10$. Seguem-se $n$ linhas: cada uma tem uma sequência de letras minúsculas que representam os

dados enviados numa mensagem. Em cada mensagem, as letras são distintas e, globalmente, não são usadas mais do que 10 letras diferentes.

### Output

Uma única linha com a lista de dados (letras únicas) que se conseguiram descodificar separados por vírgulas, e um espaço a seguir a cada vírgula, e ordenados alfabeticamente. Caso não se consiga descodificar nada deve ter simplesmente a palavra `erro`.

### Exemplo 1

#### Input

```
2
pr
r
```

#### Output

```
p, r
```

### Exemplo 2

#### Input

```
4
abc
ab
cd
ac
```

#### Output

```
a, b, c, d
```

### Exemplo 3

#### Input

```
3
abc
abd
de
```

#### Output

```
erro
```

### Exemplo 4

#### Input

```
3
abc
ab
dc
```

#### Output

```
c, d
```

**Nota:** O Exemplo 1, diz respeito ao nó recetor que está à direita na figura.