15ª Maratona de Programação

11 de abril de 2015

Caderno de Problemas

Regras: ACM *International Collegiate Programming Contest*

Brasil – **Desafio ALGAR TELECOM**

(Este caderno contém 10 problemas; as páginas estão numeradas de 1 a 12)

Problema A
Mostre a Sequência

Arquivo fonte: sequencia.c, sequencia.cpp ou sequencia.java

O problema de encontrar o próximo termo de uma sequência de números é geralmente proposta em testes de QI. Sua tarefa é gerar os N termos de uma sequência a partir de uma codificação desta sequência.

Considere que $S = (S_i)$, $i \in \mathbb{N}$, descreva uma sequência de números reais onde o i-ésimo elemento é o S_i. Uma sequência constante é definida com o seguinte operador:

$$S = [n] \text{ onde } S_i = n \quad \forall\, i \in \mathbb{N}, \text{ onde } n \in \mathbb{Z}.$$

Outros operadores em uma determinada sequência de números $S = (S_i)$, $i \in \mathbb{N}$, são definidos da seguinte maneira:

$$V = [m + S] \quad \text{onde} \quad V_i = \begin{cases} m & , i = 1 \\ V_{i-1} + S_{i-1} & , i > 1 \end{cases}$$

$$V = [m * S] \quad \text{onde} \quad V_i = \begin{cases} m * S_1 & , i = 1 \\ V_{i-1} * S_i & , i > 1 \end{cases}$$

Onde $m \in \mathbb{N}$.

Veja alguns exemplos:

```
[2 + [1]]               = 2, 3, 4, 5, 6 ···
[1 + [2 + [1]]]         = 1, 3, 6, 10, 15, 21, 28, 36 ···
[2 * [1 + [2 + [1]]]] = 2, 6, 36, 360, 5400, 113400 ···
[2 * [5 + [- 2]]]       = 10, 30, 30, -30, 90, -450, 3150 ···
```

Sua tarefa é gerar os N termos de uma sequência, uma vez apresentada sua codificação.

Entrada

A entrada é formada de vários casos de testes. Cada caso é formado por uma única linha contendo a codificação (sem espaços), seguido de um espaço separando a codificação de um número N ($2 \leq N \leq 50$). A última linha contém o número zero indicando o fim das entradas.

Saída

Para cada caso de teste, deverá ser gerada apenas uma linha contendo a sequência de N termos separados por espaço em branco. Lembrando que o último número da sequência também deverá ser seguido de um espaço em branco.

Exemplos de Entrada

```
[2+[1]] 3
[2*[5+[-2]]] 7
0
```

Exemplos de Saída

```
2 3 4
10 30 30 -30 90 -450 3150
```

Problema B

Família Feliz

Arquivo fonte: familia.c, familia.cpp ou familia.java

Duduzinho, seu papai Dudu, sua mamãe e seu vovô foram assistir uma luta de MMA. Ao chegarem ao local do evento, papai Dudu viu vários lutadores e pessoas famosas tirando algumas fotografias com os fãs e também resolveu ir lá para fazer o mesmo. Como papai Dudu é um super fã, resolveu tirar uma fotografia com todas as pessoas famosas que lá estavam. Então ele pediu ao filho Duduzinho para tirar uma fotografia com toda a família e mais os ídolos. Para ter fotografias diferentes e sempre com todas as pessoas famosas, ele pediu para que a família ficasse sempre do lado esquerdo, e do lado direito, os famosos revezariam a posição para sair na fotografia. Todas as pessoas devem aparecer em todas as fotografias.

Sua missão será calcular o número máximo de fotografias que Duduzinho terá que fazer. O tempo máximo de execução não poderá exceder a 3 segundos.

Entrada

Haverá diversos casos de teste. Cada caso de teste é composto por um número inteiro N (1 ≤ N ≤ 10000), indicando o número de pessoas famosas.

O último caso de teste é indicado quando N = 0, o qual não deverá ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro, indicando o número de fotografias que serão realizadas.

Exemplos de Entrada

```
8
9
0
```

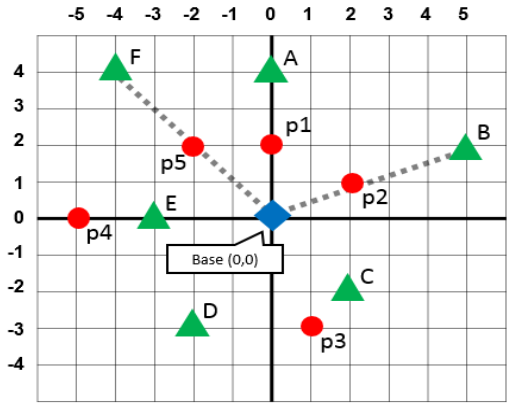
Exemplos de Saída

```
40320
362880
```

Problema C
Drone rota direta

arquivo: drone.c , drone.cpp, drone.java

O Lorde Almirante Atlan resolveu deixar sua vida aventuras e usar todo seu conhecimento cientifico para desenvolver o negócio de entregas rápidas utilizando **drones** (veículos aéreos não tripulados – VANTs) na cidade de Nova Terrânia. Parece que o negócio é promissor, mas ainda está em fase inicial. Por isso, Atlan considerou algumas restrições para as entregas. Partindo do seu quartel-general, que está na coordenada (0,0) de um plano cartesiano que mapeia a cidade de Nova Terrânia, toda entrega somente pode ser feita se o Drone voar em linha reta até o destino. Mas o **drone** pode voar no máximo até 200 metros de altura, qualquer torre, prédio ou obstáculo maior que 200 metros irá obrigar o **drone** a fazer um desvio.
Na figura a seguir apenas os destinos B, C, D, E podem ser alcançados sem desvios. A rota entre a base e os destinos A e F passa exatamente sobre um obstáculo, ponto p1 e p5. Observe que o ponto p2, embora muito próximo à rota para o destino B, não está exatamente sobre a rota, por isso não é necessário um desvio.



Entrada

A entrada é composta de vários casos de testes.

Cada caso de teste é formado de uma linha contendo dois inteiros positivos, D e P(0 ≤ D ≤ 10000, 1 ≤ P ≤ 10000) que representam, respectivamente, o número de destinos e o número de obstáculos. Porém, quando D = 0, indica o final dos casos de testes.

Cada uma das D linhas seguintes contém um par X, Y de inteiros, que representam as coordenadas do ponto de destino. Entre o X e Y existe um espaço em branco.

Cada uma das P linhas seguintes contém um par X, Y de inteiros, que representam as coordenadas de cada obstáculo. Entre o X e Y existe um espaço em branco. Considere -10000 ≤ X ≤ 10000 e -10000 ≤ Y ≤ 10000.

Saída

Para cada caso de teste, seu programa deverá imprimir os pontos de destino que não possuem nenhum obstáculo entre a base e o ponto. Cada ponto deve ser impresso em uma linha, sendo dois inteiros x, y e devem aparecer na mesma ordem que foram informados na entrada.

Exemplo de Entrada

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

Exemplo de Saída

```
5 2
2 -2
```

Problema D
Escada

arquivo: escada.c , escada.cpp, escada.java

Escadas rolantes sem dúvidas facilitam muito a vida das pessoas. Subir escadas é uma das tarefas mais tediosas já inventadas (após a invenção das escadas normais).

Após algumas observações você percebeu que há muita energia gasta com escadas rolantes, pois elas continuam funcionando mesmo quando não há ninguém utilizando. Para contornar isso, o dono de um shopping local instalou um sensor que verifica quando há alguém na escada rolante. Quando o sensor não detecta nenhuma presença, a escada rolante é desativada, assim economizando energia até que a próxima pessoa chegue.

Para ser mais específico, o sistema funciona da seguinte maneira: a escada está inicialmente desativada. O tempo necessário para que uma pessoa chegue de um lado até o outro da escada rolante é 10 segundos. Ou seja, se uma única pessoa se aproximar da escada rolante no tempo t, a escada rolante ficará ativada nos tempos t, t+1, t+2, …, t+8 e t+9, e será desativada no tempo t+10, momento no qual a pessoa já saiu da escada rolante. Tal duração pode ser prolongada caso uma ou mais pessoas se aproximem da escada rolante durante tal processo.

O dono do shopping local agora pediu sua ajuda. Escreva um algoritmo que, dados os tempos em que as pessoas se aproximaram da escada rolante, diga por quantos segundos a escada ficou ativada.

Entrada

Haverá no máximo 30 casos de teste. Cada caso de teste inicia com uma linha contendo um inteiro N, indicando o número de pessoas que usaram a escada rolante no dia em questão (1 ≤ N ≤ 100).

Na linha seguinte haverá N inteiros distintos, dados em ordem crescente, indicando o tempo t em que cada pessoa se aproximou da escada (1 ≤ t ≤ 1000).

O último caso de teste é indicado quando N = 0, o qual não deverá ser processado.

Saída

Para cada caso de teste imprima uma linha, contendo um inteiro, indicando o número de segundos que a escada rolante ficou ativa.

Exemplo de Entrada

```
1
5
2
12 25
2
13 16
5
15 20 29 31 50
0
```

Exemplo de Saída

```
10
20
13
36
```

Problema E
Ataque Programado

arquivo: ataque.c , ataque.cpp, ataque.java

Você é o líder de uma equipe de soldados de elite, e acaba de descobrir que os soldados que você enviou recentemente para atacar os postos inimigos foram capturados e mantidos como refém. Sua estratégia agora é recuperar sua tropa sem perder um soldado em batalha, e sem nunca deixar que o inimigo soe o alarme.

Existem N postos inimigos e M linhas de visão entre eles, de tal modo que se existe uma linha de visão do posto A ao posto B, os soldados do posto A saberiam quando o posto B fosse atacado e soariam o alarme. Como seu objetivo é total descrição você decidiu que só atacaria um posto quando todos os postos que tem linha de visão sobre ele tivessem sido atacados anteriormente, o que impossibilitaria que o alarme fosse soado.

Inicialmente você tem S soldados em sua tropa. Em cada posto inimigo há E soldados inimigos e F soldados reféns. Para garantir que cada ataque seja um sucesso, você decidiu que só vai atacar um posto quando o número de soldados em sua tropa for maior que o número de soldados inimigos daquele posto. Após cada ataque, os soldados reféns daquele posto são adicionados à sua tropa para os próximos ataques.

O plano parece bom, mas é preciso ter absoluta certeza de que é possível completá-lo. Com os dados sobre os postos trazidos pelo seu espião, descubra se é possível atacar todos os postos inimigos seguindo as duas restrições acima.

Entrada
Haverá no máximo 30 casos de teste. Cada caso de teste inicia com três inteiros, N, M e S, indicando o número de postos, o número de linhas de visão e o número inicial de soldados de elite em sua equipe, respectivamente ($1 \le N \le 10$, $0 \le M \le 10$, $1 \le S \le 100$).

Em seguida haverá uma linha com N inteiros e_i, onde o i-ésimo inteiro indica quantos soldados inimigos há no posto i ($1 \le e_i \le 10$, para todo $1 \le i \le N$).

Em seguida haverá uma linha com N inteiros f_i, onde o i-ésimo inteiro indica quantos soldados reféns há no posto i ($0 \le f_i \le 100$, para todo $1 \le i \le N$).

Em seguida haverá M linhas, cada uma contendo dois inteiros A e B, indicando que o posto A tem uma linha de visão sobre o posto B ($1 \le A, B \le N$, $A <> B$).

O último caso de teste é indicado quando N = M = S = 0, o qual não deverá ser processado.

Saída
Para cada caso de teste imprima uma linha, contendo a palavra “possivel” caso seja possível atacar todos os postos respeitando as restrições dadas, ou “impossivel” caso contrário.

Exemplo de Entrada
```
2 1 2
1 2
1 0
1 2
2 1 2
1 2
1 0
2 1
3 3 2
1 2 3
1 1 1
1 2
2 3
2 1
0 0 0
```
Exemplo de Saída
```
possivel
impossivel
impossivel
```

Problema F
A plantação de Gucky

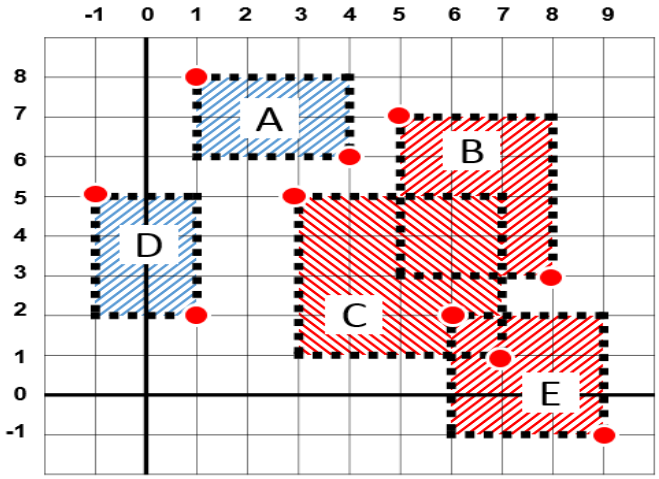
Arquivo fonte: plantacao.c, plantacao.cpp ou plantacao.java

A paixão de Gucky por cenouras só perde para sua lendária audácia. Sempre que possível Gucky vai para sua fazenda para relaxar. E por isso, resolveu cultivar as melhores cenouras do império e de todas as variedades.

Cada variedade de cenoura seria cultivada em um canteiro separado. Gucky definiu que todo canteiro teria uma forma quadrada ou retangular e sempre estaria alinhado na direção Norte-Sul para que as cenouras recebessem a mesma quantidade de luz do sol. Os canteiros possuem tamanhos variáveis de acordo com as restrições do terreno e do tipo de cultivo da cenoura. Cada canteiro tem no mínimo uma unidade de área.

Em seguida Gucky pediu a seu melhor amigo, Reginald, para demarcar os vértices de cada canteiro em um mapa. O mapa é constituído por um plano cartesiano que considera a sede da fazenda no ponto (X, Y) igual a (0, 0) e cujo eixo Y é alinhado na direção norte-sul. Portanto cada canteiro é representado por quatro pares (X Y) de inteiros. Os lados dos canteiros sempre estarão paralelos ao eixo X ou Y do mapa.

Porém, como era de se esperar, Reginald resolveu irritar Gucky. Ele fez algumas marcações onde alguns canteiros ficam sobrepostos (veja a figura os canteiros B, C, E), e ao contrário de entregar o mapa desenhado entregou apenas uma lista com as coordenadas de dois vértices opostos de cada canteiro.
Ajude Gucky a processar a lista e descobrir quais os canteiros estão sobrepostos.



Entrada
A primeira linha da entrada contém um inteiro positivo N (1≤ N ≤ 10000) representando o número de canteiros que serão processados.
Cada uma das N linhas seguintes contém dois pares de inteiros (X Y X Y), que representam as coordenadas de dois vértices de cada canteiro (-100000 ≤ X ≤ 100000, -100000 ≤ Y ≤ 100000). Entre cada inteiro existe um espaço em branco.
Os vértices de cada canteiro sempre são fornecidos na ordem: superior-esquerdo, inferior-direito.

Saída
Seu programa deve imprimir os vértices dos canteiros que ficam sobrepostos. Se algum ponto da borda de um canteiro coincide com algum ponto da borda de outro canteiro, considera-se também uma superposição.
Separando cada inteiro, existe um espaço.

Exemplo de entrada
```
5
1 8 4 6
5 7 8 3
3 5 7 1
-1 5 1 2
6 2 9 -1
```
Exemplo de saída
```
5 7 8 3
3 5 7 1
6 2 9 -1
```

Problema G

Muito fácil ?

Arquivo fonte: facil.c, facil.cpp ou facil.java

O número decimal 585 = 1001001001 (binário) é um palíndromo nas duas bases.
Sua tarefa é encontrar a soma de todos os números palíndromos na base decimal e na base binária dentro de uma faixa informada.

Observe que um número palíndromo nas duas bases não pode ter zeros à esquerda. O tempo de execução não pode exceder 2 segundos.

Entrada

A entrada é composta por vários casos de testes.
A primeira linha é composta por um inteiro N (1 ≤ N ≤ 100) indicando o número de casos de testes.
As próximas N linhas contém dois números inteiros M1(1 ≤ M1 ≤ 1000000) e M2 (1 ≤ M2 ≤ 1000000) sendo M1 ≤ M2, indicando o início e o final da faixa em que se deve calcular a soma dos números que são palíndromos nas duas bases. M1 e M2 são separados por um espaço.

Saída

Em cada linha deverá ser impresso a soma de todos os números que são palíndromos na faixa informada e também o total de números encontrados. Estes números devem ser separados por um espaço.

Exemplo de entrada

```
2
1 10
20 30
```

Exemplo de saída

```
25 5
0 0
```

Problema H
Rede Social

Arquivo fonte: redesocial.c, redesocial.cpp ou redesocial.java

Você já parou para imaginar o quanto pode ser importante descobrir quais as relações de amizade em uma rede social têm impacto direto na propagação de uma informação? Sua missão nesse problema é calcular a aresta mais valiosa de uma dada rede social. Essa medida é conhecida como *betweenness* de uma aresta. Quanto maior o *betweenness*, mais valiosa é a aresta. Veja como é fácil calcular essa medida usando o seguinte algoritmo:

1) Para cada nó X do grafo:

1.1) Faça uma busca em largura (BFS) no grafo, começando pelo nó X. Note que o nível (level) de cada nó é o tamanho do menor caminho a partir de X até o nó. As arestas entre os níveis são chamadas de arestas DAG (Directed Acyclic Graph). Cada aresta DAG será parte de no mínimo um menor caminho a partir da raiz X. Se existe uma aresta DAG (Y,Z), onde Y está no nível acima de Z, então o nó Y é chamado de pai de Z e Z é filho de Y.

1.2) A segunda etapa é colocar um *label* em cada nó do grafo BFS obtido no passo anterior. O *label* de um nó corresponde ao número de menores caminhos que existem a partir da raiz até ele. Comece rotulando a raiz com 1. Daí, utilizando a abordagem *top-down*, rotule cada nó Y como sendo a soma dos *labels* de seus pais (Fig. 2).

1.3) O último passo é calcular o crédito a ser associado a cada aresta *e* do grafo BFS. Esse cálculo é feito de baixo para cima. As regras são:

a) Cada folha no grafo BFS recebe crédito 1.

b) A uma aresta DAG *e*, chegando no nó Z a partir do nível acima é dado um crédito compartilhado de Z, proporcional à fração de menores caminhos a partir da raiz até Z que passam por *E*. Formalmente, sejam Y_1, Y_2, ..., Y_k os pais de Z. Seja p_i o número de menores caminhos a partir da raiz até Y_i. Este número foi calculado no passo 1.2. Então, o crédito da aresta (Y_i,Z) é o crédito de Z vezes p_i dividido por $\sum_{j=1}^{k} pj$

c) Cada nó que não é folha recebe crédito igual a 1 mais a soma dos créditos das arestas DAG a partir desse nó até o nível abaixo (Fig. 3).

2) Depois de calcular o crédito das arestas para cada grafo BFS computado (um grafo para cada nó como raiz), os créditos de cada aresta são somados. Como o menor caminho foi computado 2 vezes, é necessário dividir o crédito final obtido em cada aresta por 2 (Fig. 4).

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste inicia com uma linha contendo um inteiro N (1 ≤ N ≤ 50), que indica o número de linhas subsequentes. Cada uma das N linhas subsequentes contém duas *strings* separadas por espaço do tipo "*N1 N2*" (sem aspas), indicando a existência de uma aresta entre os nós *N1* e *N2*. O final da entrada é indicado por final de arquivo (EOF). Não existem arestas repetidas na entrada.

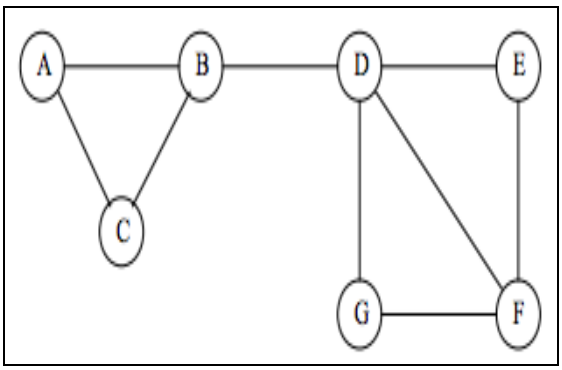


Fig.1: Grafo de exemplo

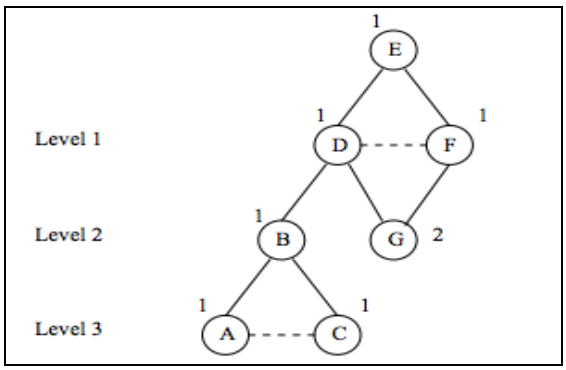


Fig.2: Exemplo após passo 1.2 para nó E como raiz

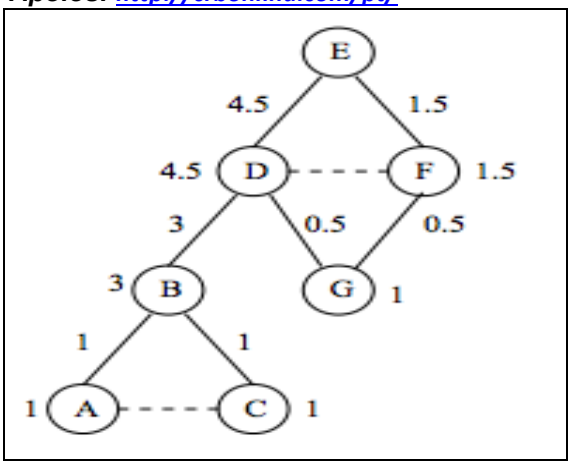


Fig. 3: Exemplo após passo 1.2 para nó E como raiz

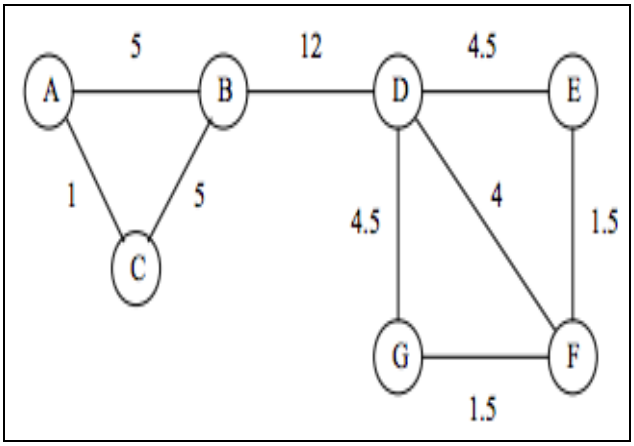


Fig. 4: Resultado final

Saída

Imprima uma linha contendo a aresta com maior *betweenness*. Considere que o maior *betweenness* é único e não haverá empate entre duas arestas. Considere também que, apesar de ser um grafo não dirigido, a aresta com maior *betweenness* deve ser impressa como foi descrita na entrada.

Exemplo de entrada

```
9
a b
a c
b c
b d
d e
d f
d g
e f
f g
```

Exemplo de saída

```
b d
```

Problema I

Léxico

Arquivo fonte: lexico.c, lexico.cpp ou lexico.java

O mundo está ficando mesmo muito moderno! Você sabia que é possível prever o resultado das eleições de um país apenas analisando o sentimento dos *tweets* publicados durante a campanha eleitoral?

Um *tweet* é um pequeno texto de no máximo 140 caracteres que um usuário pode publicar. Analisar o sentimento de um tweet significa dizer se ele é *positivo*, *negativo* ou *neutro*. Por exemplo, o tweet "*Adoro música clássica! #mozart*" é um tweet com sentimento positivo. Por outro lado, o seguinte tweet expressa um sentimento negativo: "*O PS4 é muito ruim" #ps4fake*". O sentimento neutro existe quando não é possível identificar se o tweet é positivo ou negativo.

Para identificar o sentimento de um tweet deve-se considerar o dicionário de termos abaixo e as seguintes regras:

Positivo	**Negativo**
excelente	ruim
bom	pessimo
otimo	odeio
adoro	droga
amo	fraco

1) O tweet é *positivo* se a maioria dos termos que são iguais aos do dicionário é positiva
2) O tweet é *negativo* se a maioria dos termos que são iguais aos do dicionário é negativa
3) Se para cada termo do dicionário que aparece no tweet existe a palavra "nao" (sem aspas) imediatamente antes dele (ex.: *"nao amo", "nao odeio"*), o sentimento é invertido em relação ao dicionário e a expressão é tratada como um termo de sentimento invertido
4) Para qualquer outro caso, o sentimento é neutro

Neste problema sua missão é descobrir se um conjunto de tweets tem em sua maioria tweets positivos, negativos ou neutros.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste indica um conjunto de tweets sobre um determinado tema e inicia com uma linha contendo um inteiro N ($1 \leq N \leq 50$), indicando a quantidade de tweets. Cada uma das N linhas subsequentes contém um tweet formado por palavras minúsculas sem acentuação. **O final da entrada é indicado por final de arquivo (EOF)**.

Saída

Para cada caso de teste imprima uma linha contendo o sentimento predominante no conjunto de tweets daquele teste – *positivo*, *negativo* ou *neutro*. Considere que sempre haverá um sentimento predominante.

Exemplo de Entrada

```
3
este filme eh otimo! adoro! #interestelar
o filme interestelar eh bom
#interestelar achei muito fraco
4
nao odeio o presidente obama #voteforobama
nao adoro republicanos, acho pessimo defender aborto
amo o obama, mas o governador eh fraco
o atual governo eh excelente #obamaeua
```

Exemplo de Saída

```
positivo
positivo
```

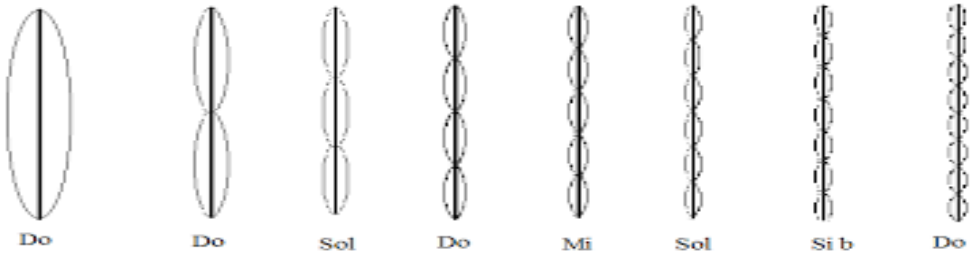
Problema J
Série Harmônica

Arquivo fonte: harmonica.c, harmonica.cpp ou harmonica.java

Quando uma nota é emitida por um instrumento musical, ou mesmo pela voz humana, pensamos ouvir apenas uma única nota musical, mas na verdade uma nota não está soando sozinha. Ele vai gerar toda uma série de sons menos audíveis que irão compor o som; a isso damos o nome de Série Harmônica.

O estudo e entendimento da Série Harmônica teve início no século VI antes de Cristo com as experiências feitas pelo filósofo e matemático grego Pitágoras. Por suas descobertas é possível estabelecer uma relação direta entre melodia e harmonia, sendo que seus conceitos e definições são utilizados até os dias atuais (oitavas, ciclo de quintas, etc).

A série harmônica é fisicamente infinita e, dado o valor (Hz) da primeira frequência, ou a fundamental, o próximo harmônico, ou segundo harmônico, será duas vezes maior que a fundamental (som mais agudo). O terceiro harmônico será 3 vezes maior que a fundamental, e assim por diante. Em outras palavras, na série harmônica as frequências são `f, 2*f, 3*f , 4*f ,..., isto é, f= n*f`



O 2o harmônico de uma fundamental é a mesma nota só que uma oitava acima (mais aguda). Isso significa que uma oitava acima de Dó, por exemplo, também é uma nota Dó, só que mais aguda (**Dó**, Ré, Mi, Fá, Sol, Lá, Si, **Dó**) O 3o harmônico de uma fundamental também especialmente importante pois é o 5o grau da fundamental. Sendo assim, se a fundamental for a nota Dó, o 2o harmônico será Dó uma oitava acima e o 3o harmônico será a nota Sol (De acordo com a sequência **Dó**, Ré, Mi, Fá, **Sol**, Lá, Si).

Dó	Ré	Mi	Fá	Sol	Lá	Si	Dó
1º	2º	3º	4º	5º	6º	7º	8ª

As notas da série harmônica definidas por Pitágoras deram origem às chamadas escalas musicais, no chamado ciclo das quintas (tomando o 5o grau de cada nota sucessivamente). Você deverá calcular as notas musicais (valores em Hz) da escala pentatônica (5 notas).

Entrada

A entrada é composta por vários casos de teste. A primeira linha da entrada é composta por um inteiro N(1≤ N ≤ 50) que indica o número de casos de testes. As próximas N linhas, contém um número real R(1≤ N ≤ 1000) indicando a nota fundamental(frequência em Hz) a ser analisada. O número R contém apenas duas casas decimais.

Saída

Para cada caso de teste imprima uma linha contendo as 5 notas(frequência em Hz) da escala pentatônica. Os valores calculados devem ter apenas duas casas decimais truncadas, devem ser separados por um espaço inclusive após o último valor.

Exemplo de entrada

```
110
440
265.50
```

Exemplo de saída

```
110.00 165.00 123.75 185.62 139.21
440.00 660.00 495.00 742.50 556.87
265.50 398.25 298.68 448.03 336.02
```