Maia

2^a Seletiva Interna – 2010/2

Sevidor BOCA:

`http://192.168.0.21:8001`
(acesso interno)

`http://200.19.106.219:8001`
(acesso externo)

Organizadores:

Alexandre Gonçalves Silva, Roberto Silvio Ubertino Rosso Jr., Claudio Cesar de Sá
{*alexandre,rosso,claudio*} *at joinville dot udesc dot br*

Lembretes:

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa. Deve-se considerar entradas e saídas padrão.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido ...), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos.
- Utilize o *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos.

Maia

2^a Seletiva Interna da UDESC

28 de agosto de 2010

Conteúdo

1 Problema A: Reserva em hotéis

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

Uma empresa de transportes frequentemente precisa entregar mercadorias de uma cidade para outra. A empresa de transportes tem feito um acordo especial com uma rede de hotéis que permite aos seus motoristas ficar nos hotéis da rede de graça. Motoristas são autorizados a dirigir por até 10 horas por dia.

A empresa de transportes quer encontrar uma rota a partir da cidade de partida para a cidade de destino tal que um motorista possa sempre passar a noite em um dos hotéis da rede, e que ele precise dirigir, no máximo, 10 horas de um hotel para o próximo hotel (ou destino).

Naturalmente, o número de dias necessários para entregar a mercadoria também deve ser minimizado.

Especificação da entrada

A entrada contém vários casos de teste. Cada caso de teste inicia com uma linha contendo um inteiro n, $(2 \leq n \leq 10000)$, o número de cidades a serem consideradas no planejamento da rota.

Para simplificar, as cidades são numeradas de 1 a n, onde 1 é a cidade de partida, e n é a cidade de destino. A próxima linha contém um inteiro h seguido pelos números c_1, c_2, ..., c_h, indicando o número de cidades onde os hotéis da rede estão localizados. Você pode assumir que $0 \leq h \leq \min(n, 100)$.

A terceira linha de cada caso de teste contém um inteiro m $(1 \leq m \leq 10^5)$, o número de estradas a serem consideradas para o planejamento da rota.

As m linhas seguintes descrevem as estradas. Cada estrada é descrita por uma linha com 3 inteiros a, b, t $(1 \leq a, b \leq n$ e $1 \leq t \leq 600)$, onde a, b, são as duas cidades ligadas por estrada, e t é o tempo, em minutos, necessários para o motorista dirigir de uma extremidade da estrada a outra.

A entrada é finalizada por $n = 0$.

Especificação da saída

Para cada caso de teste, imprima uma linha com o número mínimo de hotéis que a empresa de transportes precisa reservar para a entrega a partir de uma cidade 1 para a cidade n.

Se não for possível encontrar uma rota tal que o motorista dirija por, no máximo, 10 horas por dia, imprima -1.

Exemplo de entrada

```
6
3 2 5 3
8
1 2 400
3 2 80
3 4 301
4 5 290
5 6 139
1 3 375
2 5 462
4 6 300
3
0
2
1 2 371
2 3 230
0
```

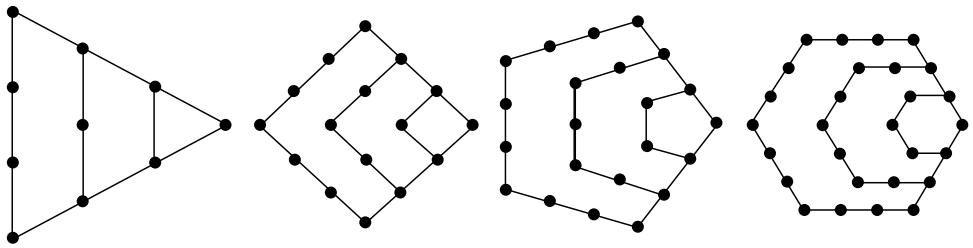
Exemplo de saída

```
2
-1
```

2 Problema B: Números poli-poligonais

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

Um número *poligonal* é um número que pode ser representado por um arranjo geométrico regular de pontos igualmente espaçados, onde o arranjo constitui um polígono regular. Alguns exemplos são mostrados nas figuras abaixo.



A primeira figura mostra os primeiros 4 números *triangulares* 1, 3, 6, 10. Os três seguintes mostram os primeiros quatro números *quadrados*, *pentagonais* e *hexagonais*, respectivamente. Em geral, os números *k-gonal* são aqueles cujos pontos definem um k-gon regular (daí números triangulares serem 3-gonal, números quadrados serem 4-gonal, etc.) Definiremos k como um *índice* do número poligonal. Para este problema, você deve encontrar números que são k-gonal para dois ou mais valores de k. Vamos chamar estes números de *poli-poligonal*.

Especificação da entrada

A entrada é composta por várias instâncias problema. Cada instância consiste em 3 linhas. A primeira linha é um inteiro não-negativo $n \leq 50$, indicando o número de tipos de números poligonais de interesse neste problema. Note que esta linha pode conter mais de 80 caracteres. A próxima linha contém n inteiros, indicando os índices destes números poligonais (todos distintos e em ordem crescente). Por exemplo, se a primeira linha contiver o valor `3`, e na linha seguinte, os valores `3 6 10`, então esta instância do problema estaria interessada em números 3-gonal, 6-gonal e 10-gonal números. Cada índice k estará no intervalo $3 \leq k \leq 1000$. A última linha da instância do problema contém um único inteiro positivo $s \leq 10000$, que serve como ponto de partida para a busca de números de poli-poligonal. Um valor de $n = 0$ finaliza a entrada.

Especificação da saída

Para cada instância do problema, você deve produzir os 5 números poli-poligonal seguintes maiores ou igual a s. Cada número deve estar em uma única linha e em conformidade com o seguinte formato:

```
num:k1 k2 k3 ...
```

onde `num` é o número de poli-poligonal, e `k1`, `k2`, `k3` ... são os índices (em ordem crescente) do número poli-poligonal igual a `num`. Um único espaço deve separar cada índice, e você deve separar cada instância do problema com uma única linha em branco. A entrada dos juízes será tal que o valor máximo para qualquer número poli-poligonal poderá ser representado por uma variável **long**.

Exemplo de entrada

```
10
6 7 8 9 10 11 12 13 14 15
1000
5
3 4 13 36 124
1
0
```

Exemplo de saída

```
1216:9 12
1540:6 10
1701:10 13
2300:11 14
3025:12 15

1:3 4 13 36 124
36:3 4 13 36
105:3 36
171:3 13
1225:3 4 124
```

3 Problema C: k-inter

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

Chico dispõe de conjuntos de valores inteiros, sendo cada um destes inteiros **limitado** entre 0 e 100 (inclusive estes dois valores). Em um de seus experimentos estatísticos, ele pretende substituir todos os elementos do conjunto por um único valor denominado de k-inter. O motivo é que seus conjuntos possuem, em geral, alta cardinalidade (ou seja, têm normalmente expressiva quantidade de números) e ocupam muita memória.

k-inter é um valor inteiro, representativo para seus propósitos, e consiste na média entre k valores posicionados exatamente no meio da sequencia ordenada dos valores do conjunto. Esta média, que deve ser arredondada para o inteiro mais próximo, vai ser determinada por Chico da forma mais eficiente (rápida) possível. Veja um exemplo a seguir para $N = 9$ valores e $k = 5$:

$$[3,0,3,0,1,2,3,2,4] \;\Rightarrow\; [0,0,\underbrace{1,2,2,3,3}_{k=5},3,4]$$

$$\text{arredonda}(\ (1+2+2+3+3)\ /\ 5\ ) = 2$$

Entrada

A entrada contém vários casos de teste. Cada teste é formado por um inteiro N ($k < N \leq 125000$) e o valor k ($0 < k \leq 15$) na primeira linha. N valores inteiros vêm na linha seguinte com o conteúdo do conjunto. Considere N e k sempre ímpares. Na última linha há dois zeros, significando a finalização da entrada.

Saída

Para cada caso de teste, deve-se produzir uma linha com a média dos k valores do meio do conjunto ordenado. A média deve ser arredondada para o inteiro mais próximo.

Exemplo de entrada

```
9 5
3 0 3 0 1 2 3 2 4
5 1
3 2 5 3 0
5 3
1 1 1 0 4
5 3
2 4 0 1 2
9 3
5 4 4 4 1 3 5 8 1
0 0
```

Exemplo de saída

```
2
3
1
2
4
```

4 Problema D: Eldorado

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

Bruce Force foi para Las Vegas, o Eldorado para os jogadores. Ele está interessado principalmente em um jogo de apostas, onde uma máquina faz uma sequência de n números por meio de valores aleatórios. Cada jogador deve estimar com antecedência, quantas subsequencias crescentes de comprimento k vai existir na sequência de números.

Uma subsequência de uma sequência $a_1, \ldots, a_n$ é definida como $a_{i_1}, \ldots, a_{i_l}$, onde $1 \leq i_1 < i_2 < \ldots < i_l \leq n$. A subsequência é crescente, se $a_{i_{j-1}} < a_{i_j}$ para todos os $1 < j \leq l$.

Bruce não confia no Casino para contar corretamente o número de subsequências crescentes de comprimento k. Ele perguntou se você não poderia resolver este problema para ele.

Especificação da entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois números n e k ($1 \leq k \leq n \leq 100$), onde n é o comprimento da sequência estabelecida pela máquina, e k é o comprimento desejado das subsequências crescentes. A linha a seguir contém n pares inteiros distintos a_i ($-10000 \leq a_i \leq 10000$), onde a_i é o i-ésimo número na sequência estabelecida pela máquina.

O último caso de teste é seguido por uma linha contendo dois zeros.

Especificação da saída

Para cada caso de teste, imprima uma linha com o número de subsequências crescentes de comprimento k que a sequência de entrada contém. Você pode assumir que as entradas são escolhidas de modo que este número seja representado por um inteiro com sinal de 64 bits (em C/C++, você pode usar o tipo de dado "long long" e, em Java, o tipo "long").

Exemplo de entrada

```
10 5
1 2 3 4 5 6 7 8 9 10
3 2
3 2 1
4 1
-3269 5137 8673 7238
0 0
```

Exemplo de saída

```
252
0
4
```

5 Problema E: Pintura em preto e branco

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

Você está visitando um museu que contém uma série de pinturas modernas. Em particular, você percebe uma pintura contendo apenas quadrados pretos e brancos, dispostos em linhas e colunas como em um tabuleiro de xadrez (não há dois quadrados adjacentes com a mesma cor).

Você entediado se pergunta quantos tabuleiros 8×8 de xadrez podem ser representados dentro desta pintura. O canto inferior direito de um tabuleiro de xadrez deve ser sempre branco.

Especificação da entrada

A entrada contém vários casos de teste. Cada caso de teste consiste de uma linha com três inteiros n, m e c. ($8 \leq n, m \leq 40000$), onde n é o número de linhas da pintura, e m é o número de colunas da pintura. c é sempre 0 ou 1, onde 0 indica que o canto inferior direito da pintura é preto, e 1 indica que este canto é branco.

O último caso de teste é seguido por uma linha com três zeros.

Especificação da saída

Para cada caso de teste, imprima o número de tabuleiros de xadrez inseridos na pintura dada.

Exemplo de entrada

```
8 8 0
8 8 1
9 9 1
40000 39999 0
0 0 0
```

Exemplo de saída

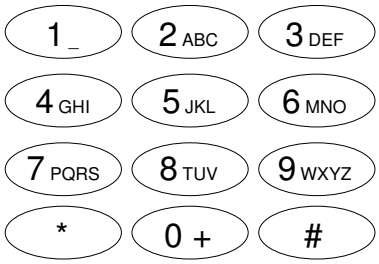
```
0
1
2
799700028
```

6 Problema F: Teclado de celular

`Arquivo: t9.[c|cpp|java]`

A um tempo atrás, era muito complicado criar uma mensagem de *Short Message Service* (SMS) no celular. Pelo fato de você só ter nove teclas e o alfabeto ter mais do que nove letras, a maioria dos caracteres só poderia ser entrada, pressionando uma tecla várias vezes. Por exemplo, se você quiser digitar "`hello`", você teria de pressionar a tecla 4 duas vezes, a tecla 3 duas vezes, a tecla 5 três vezes, novamente a tecla 5 três vezes e, finalmente, a tecla 6 três vezes. Este procedimento é muito tedioso e impede muitas pessoas de usar o SMS.

Isto levou os fabricantes de celulares a tentar encontrar uma maneira mais fácil de digitar um texto em um celular. A solução que eles desenvolveram é chamada de entrada de texto $T9$. O "9" no nome significa que você pode digitar qualquer palavra com apenas nove teclas e sem pressioná-las mais de uma vez para cada caracter. A ideia da solução é que você simplesmente inicie a digitação das teclas sem repetição, e o software utilize um dicionário incorporado para olhar para a palavra "mais provável" correspondente à entrada. Por exemplo, para entrar "`hello`" bastaria pressionar as teclas 4, 3, 5, 5 e 6 uma única vez. Naturalmente, isto também poderia ser a entrada para a palavra "`gdjjm`", mas não sendo esta uma palavra sensata em inglês, ela pode ser ignorada. Ao excluir todas as outras "improváveis" soluções de modo a ficar apenas com palavras apropriadas em inglês, este método pode acelerar consideravelmente a escrita de mensagens curtas. Claro que, se a palavra não consta do dicionário (como um nome próprio), então ela deve ser digitada manualmente usando novamente a repetição de teclas.



Teclas de um telefone celular

Mais precisamente, a cada caracter digitado, o celular irá mostrar a combinação mais provável de caracteres que tenha encontrado até aquele ponto. Vamos supor que o celular conheça as palavras "`idea`" e "`hello`", com "`idea`" ocorrendo com mais frequência. Pressionando as teclas 4, 3, 5, 5 e 6, uma após a outra, o celular apresenta "`i`", "`id`" e, em seguida, muda para "`hel`", "`hell`" e, finalmente, mostra "`hello`".

Escreva uma aplicação para entrada de texto $T9$, que ofereça a combinação de caracteres mais prováveis depois de cada pressionamento de teclas. A probabilidade de uma combinação de caracteres é definida pela soma das probabilidades de todas as palavras no dicionário que comecem com esta combinação de caracteres. Por exemplo, se o dicionário contiver três palavras "`hell`", "`hello`" e "`hellfire`", a probabilidade de combinação de caracteres "`hell`" é a soma das probabilidades dessas palavras. Se algumas combinações têm a mesma probabilidade, seu programa selecionará a primeira em ordem alfabética. O usuário também deve ser capaz de digitar o início das palavras. Por exemplo, se a palavra "`hello`" está no dicionário, o usuário também pode inserir a palavra "`he`", pressionando as teclas 4 e 3, mesmo que esta palavra não conste no dicionário.

Especificação da entrada

A primeira linha contém o número de cenários. Cada cenário inicia com uma linha contendo o número w de palavras distintas no dicionário ($0 \leq w \leq 1000$). Estas palavras são dadas nas w linhas seguintes em ordem ordem alfabética crescente. Cada linha começa com a palavra que é uma sequência de letras minúsculas do alfabeto sem espaços em branco, seguido por um espaço e um inteiro p, $1 \leq p \leq 100$, representando a probabilidade da palavra. Nenhuma palavra conterá mais de 100 letras. Na sequência do dicionário, há uma linha contendo um único inteiro m. Seguem m linhas, cada uma composta por uma sequência de até 100 dígitos decimais $2 - 9$, seguido por um único 1 significando a "próxima palavra".

Especificação da saída

A saída para cada cenário inicia com uma linha contendo "`Scenario #i:`", onde `i` é o número do cenário a partir de 1. Para cada sequência de números s do cenário, imprima uma linha para cada pressionamento de tecla armazenado em s, exceto para o 1 no final. Nesta linha, imprima o prefix de palavra mais provável definido pelas probabilidades no dicionário e as regras de selecção $T9$ explicadas acima. Quando nenhuma das palavras no dicionário casar com a sequência numérica dada, imprima "MANUALLY" em vez de um prefixo. Finalize a saída para cada sequência de números com uma linha em branco e imprima uma linha em branco adicional ao final de cada cenário.

Exemplo de entrada

```
2
5
hell 3
hello 4
idea 8
next 8
super 3
2
435561
43321
7
another 5
contest 6
follow 3
give 13
integer 6
new 14
program 4
5
77647261
6391
4681
26684371
77771
```

Exemplo de saída

```
Scenario #1:
i
id
hel
hell
hello

i
id
ide
idea


Scenario #2:
p
pr
pro
prog
progr
progra
program

n
ne
new

g
in
int

c
co
con
cont
anoth
anothe
another

p
pr
MANUALLY
MANUALLY
```

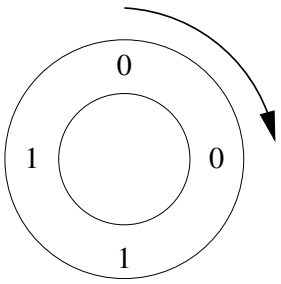
7 Problema G: Ouroboros

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

Ouroboros é uma serpente mítica do antigo Egito. Ela tem sua cauda dentro de sua boca e devora-se continuamente.

Os números de Ouroboros são números binários de 2^n bits que têm a propriedade de "geração" de todo o conjunto de números de 0 a $2^n - 1$. A geração funciona da seguinte forma: dado um número de Ouroboros, nós colocamos seus 2^n bits distribuídos em um círculo. Então, nós tomamos 2^n grupos de n bits, iniciando, a cada vez, com o bit seguinte no círculo. Esses círculos são chamados de *círculos de Ouroboros* para o número n. Vamos trabalhar somente com o menor número de Ouroboros para cada n.

Por exemplo, para $n = 2$, existem apenas quatro números de Ouroboros. São estes, 0011, 0110, 1100 e 1001. Neste caso, o menor deles é 0011. Para 0011, o diagrama e a tabela a seguir mostram o processo de encontrar todas as sequências de bits:



k	00110011...	*o(n,k)*
0	00	0
1	01	1
2	11	3
3	10	2

A tabela descreve a função $o(n, k)$ que calcula o número de k-ésimo número no círculo de Ouroboros do menor número de Ouroboros de tamanho n. Esta é a função que seu programa deve calcular.

Especificação da entrada

A entrada consiste de vários casos de teste. Para cada caso de teste, haverá uma linha contendo dois inteiros n e k ($1 \leq n \leq 15; 0 \leq k < 2^n$). O fim do arquivo de entrada é indicado por uma linha contendo dois zeros. Não processar esta última linha.

Especificação da saída

Para cada caso de teste, você deve calcular a função $o(n, k)$ e imprimir o resultado em uma linha.

Exemplo de entrada

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

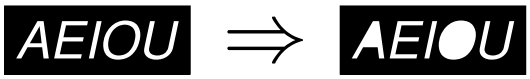
Exemplo de saída

```
0
1
3
2
```

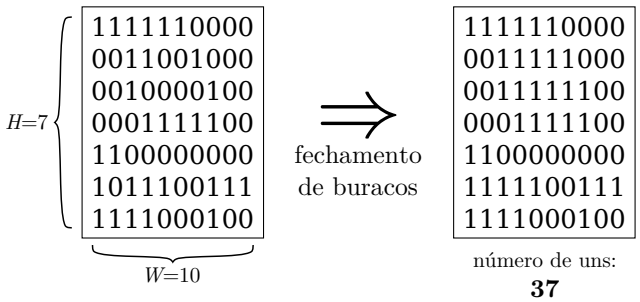
8 Problema H: Fechamento de buraco

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

Um pesquisador está desenvolvendo seu próprio reconhecedor ótico de caracteres. Ele está levantando alguns informações que podem ser utilizadas para distinguir as letras maiúsculas do alfabeto entre si. Pensou então em detectar áreas pretas totalmente cercadas por áreas brancas, no que chamou de buracos. Para isto, em um primeiro momento, imaginou um algoritmo para preencher tais buracos de branco, conforme a ilustração a seguir:



Ajude o pesquisador na implementação desta ideia, considerando a imagem tratada como uma matriz bidimensional binária, na qual, o preto é igual a 0 e o branco, igual a 1. Perceba, no exemplo, que uma região branca pode ser formada pela composição de uma única linha, coluna ou diagonal de valores iguais a 1:



Especificação da entrada

A entrada inicia com o número de casos de teste T ($1 \leq T \leq 100$). Para cada teste, a primeira linha contém dois inteiros H e W com o número de linhas e de colunas, respectivamente, da matriz ($1 \leq H, W \leq 512$). As H linhas seguintes referem-se à matriz binária.

Especificação da saída

Para cada caso de teste, deve-se produzir uma linha com o número de valores iguais 1 (um) da matriz resultante do fechamento de buracos.

Exemplo de entrada

```
1
4 6
001111
011001
010010
011100
```

Exemplo de saída

```
16
```

9 Problema I: Esquerda, Volver!

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

Este ano o sargento está tendo mais trabalho do que de costume para treinar os recrutas. Um deles é muito atrapalhado, e de vez em quando faz tudo errado – por exemplo, ao invés de virar à direita quando comandado, vira à esquerda, causando grande confusão no batalhão.

O sargento tem fama de durão e não vai deixar o recruta em paz enquanto este não aprender a executar corretamente os comandos. No sábado à tarde, enquanto todos os outros recrutas estão de folga, ele obrigou o recruta a fazer um treinamento extra. Com o recruta marchando parado no mesmo lugar, o sargento emitiu uma série de comandos "esquerda volver!" e "direita volver!". A cada comando, o recruta deve girar sobre o mesmo ponto e dar um quarto de volta na direção correspondente ao comando. Por exemplo, se o recruta está inicialmente com o rosto voltado para a direção norte, após um comando de "esquerda volver!" ele deve ficar com o rosto voltado para a direção oeste. Se o recruta está inicialmente com o rosto voltado para o leste, após um comando "direita, volver!" ele deve ter o rosto voltado para o sul.

No entanto, durante o treinamento, em que o recruta tinha inicialmente o rosto voltado para o norte, o sargento emitiu uma série tão extensa de comandos, e tão rapidamente, que até ele ficou confuso, e não sabe mais para qual direção o recruta deve ter seu rosto voltado após executar todos os comandos. Você pode ajudar o sargento?

Especificação da entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro N que indica o número de comandos emitidos pelo sargento ($1 \leq N \leq 1000$). A segunda linha contém N caracteres, descrevendo a série de comandos emitidos pelo sargento. Cada comando é representado por uma letra: '`E`' (para "esquerda, volver!") e '`D`' (para "direita, volver!"). O final da entrada é indicado por $N = 0$.

Especificação da saída

Para cada caso de teste da entrada, seu programa deve produzir uma única linha da saída, indicando a direção para a qual o recruta deve ter sua face voltada após executar a série de comandos, considerando que no início o recruta tem a face voltada para o norte. A linha deve conter uma letra entre '`N`', '`L`', '`S`' e '`O`', representando respectivamente as direções norte, leste, sul e oeste.

Exemplo de entrada

```
3
DDE
2
EE
0
```

Exemplo de saída

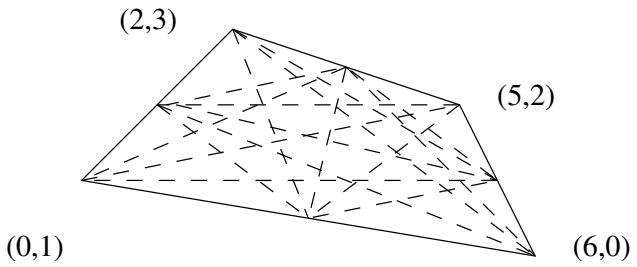
```
L
S
```

10 Problema J: This takes the cake

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

In the kingdom of Polygonia the royal family consists of the king, the queen, and the 10-year-old twins, Prince Obtuse and Prince Trisect. The twins are fiercely competitive, and on their birthday they always vie with each other for the biggest portion of the cake. The wise king and queen have devised the following way to prevent squabbles over the cake. One prince is allowed to cut the cake into two pieces, then the other prince gets to choose which of the two pieces he wants.

Cakes in Polygonia are always in the shape of a convex quadrilateral (a four-sided polygon with each internal angle less than 180 degrees). Furthermore, local custom dictates that all cake cutting must be done using a straight cut that joins two vertices, or two midpoints of the sides of the cake, or a vertex and a midpoint. For instance, the following figure shows all the possible legal cuts in a typical cake.



Your problem is to determine, for a number of different cakes, the best cut, i.e., the one that divides the cake into two pieces whose areas (we are disregarding the thickness of the cake) are as nearly equal as possible. For instance, given a cake whose vertices (when the cake is viewed from above) are located, in counterclockwise order, at the points $(0, 1)$, $(6, 0)$, $(5, 2)$ and $(2, 3)$, the best possible cut would divide the cake into two pieces, one with area 4.375, the other with area 5.125; the cut joins the points $(1, 2)$ and $(5.5, 1)$ (the midpoints of two of the sides).

Input specification

Input consists of a sequence of test cases, each consisting of four (x, y) values giving the counterclockwise traversal of the cake's vertices as viewed from directly above the cake; the final test case is followed by a line containing eight zeros. No three points will be collinear, all quadrilaterals are convex, and all coordinates will have absolute values of 10000 or less.

Output specification

For each cake, the cake number followed by the two areas, smaller first, to three decimal places of precision.

Input example

```
0 1 6 0 5 2 2 3
0 0 100 0 100 100 0 100
0 0 0 0 0 0 0 0
```

Output example

```
Cake 1: 4.375 5.125
Cake 2: 5000.000 5000.000
```