Sarah Menezes

3^a Seletiva Interna – 2012/2

Sevidor BOCA:

`http://10.20.107.205/boca/`
(acesso interno)

`http://200.19.107.205/boca/`
(acesso externo)

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Lucas Negri (coordenação técnica), Yuri Kaszubowski Lopes (coordenação técnica), Alexandre Gonçalves Silva , Roberto Silvio Ubertino Rosso Jr., André Luiz Guedes (Dinf/UFPr – revisão técnica), Fernando Deeke Sasse

Lembretes:

- Aos *javaneiros*: **o nome da classe deve ser o mesmo nome do arquivo a ser submetido**. Ex: **classe petrus**, nome do arquivo **petrus.java**;
- 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;
- A interface KDE está disponível nas máquinas Linux, que pode ser utilizada ao invés da Unity. Para isto, basta dar *logout*, e selecionar a interface KDE. Usuário e senha: *udesc*;
- O nome *Sarah Menezes* é uma homenagem a primeira judoca brasileira a ganhar uma medalha de ouro nas olímpiadas de Londres.

Patrocinador e Agradecimentos

- YoungArts – Patrocinador oficial do ano de 2012;
- DCC/UDESC;
- Rutes, Zanatta e Weskley pelo empenho neste ano;
- Alguns, muitos outros anônimos.

Sarah Menezes

3^a Seletiva Interna da UDESC

17 de agosto de 2012

Conteúdo

1 Problema A: Faça-o!

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

O Problema

Você é o chefe de uma pequena empresa de luminárias com n trabalhadores. Inspirado no personagem de Ben Stiller em *Starsky & Hutch - Justiça em Dobro*, você recentemente está com o hábito de dizer aos seus funcionários: "faça-o!" quando você quer as coisas prontas. Enquanto n_+ dos n funcionários respondem positivamente ao seus "faça-o!", n_- respondem negativamente, e n_0 são neutros as suas palavras.

No tempo 0, cada um de seus funcionários inicia o trabalho sozinho na construção de uma luminária. Cada luminária demanda 100 unidades de trabalho até o seu término. Normalmente, cada um de seus funcionários contribui com r unidades de trabalho para finalizar sua luminária, durante cada intervalo de tempo (ou a quantidade de unidades de trabalho restantes para a luminária, o que for menor). Portanto, um funcionário normalmente levaria $\lceil 100/r \rceil$ intervalos de tempo para terminar a sua luminária. Durante um intervalo, no entanto, se você gritar "faça-o!" na comunicação da empresa, funcionários, que respondem positivamente ao seu comando, farão $r + 2$ unidades de trabalho durante esse intervalo de tempo, enquanto aqueles que, respondem negativamente, farão $r - 1$ unidades de trabalho durante o intervalo de tempo.

Supondo que cada empregado trabalhe apenas na sua luminária, e assumindo que você grite "faça-o!" no máximo uma vez durante cada intervalo de tempo, seu objetivo é planejar uma sequência de "faça-o!" de modo a garantir que a soma dos tempos necessários para a finalização de todas as n luminárias seja minimizada.

Entrada

A entrada conterá múltiplos casos de teste. Cada caso de teste de entrada será dado por uma linha contendo quatro números inteiros, n_+, n_-, n_0 e r (onde $0 \leq n_+, n_-, n_0 \leq 1000$ e $1 \leq r \leq 100$). O fim da entrada é marcada por um caso de teste com $n_+ = n_- = n_0 = r = 0$ e não deve ser processado. Por exemplo:

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

Saída

Para cada caso de entrada, o programa deve imprimir a soma mínima de tempo necessário ao término de todas as n luminárias. Por exemplo:

```
188
200
```

No primeiro caso de teste, uma estratégia ótima (ideal) é gritar "faça-o!" em cada um dos 25 primeiros intervalos de tempo. Como resultado, os 3 funcionários, que respondem positivamente, provêm 4 unidades de trabalho por intervalo de tempo e, portanto, concluem suas luminárias em 25 unidades de tempo. O único funcionário que responde negativamente proporciona 1 unidade de trabalho por intervalo de tempo para os primeiros 25 intervalos de

tempo. Na sequência, ele provê 2 unidades de trabalho por intervalo de tempo e, deste modo, vai terminar em $25 + 38 = 63$ unidades de tempo. Finalmente, o funcionário neutro sempre prove 2 unidades de trabalho por intervalo de tempo e, portanto, terminará em 50 unidades de tempo. Isto dá um total de $3(25) + 63 + 50 = 188$ unidades de tempo. No segundo caso de teste, uma estratégia ótima é nunca gritar "faça-o!". Assim, todos os quatro funcionários terminariam em 50 unidades de tempo, visto que a quantidade total de tempo gasto é de 200 unidades de tempo.

2 Problema B: Lagarta

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

Um grafo não-orientado é chamado de *lagarta* se ele é conexo, não tem ciclos, e existe um caminho no grafo onde cada nó ou está neste caminho, ou é um vizinho de um nó neste caminho. Isto é, para ser uma *lagarta*, todo nó está num ramo principal ou caminho, ou é um vizinho imediato a este, em uma aresta de distância. Este caminho é chamado de **espinha da lagarta** e a espinha pode não ser única. Vocês devem verificar grafos e ver se eles são lagartas. Por exemplo, o grafo abaixo a esquerda não é uma lagarta, mas o da direita o é. Uma possível espinha é mostrada por pontos.

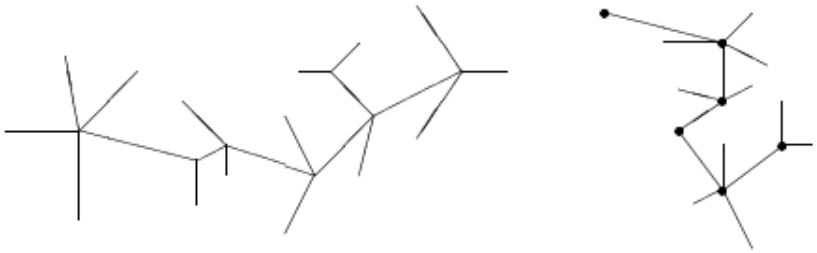


Figura 1: Exemplos de grafo do tipo *lagarta*

Entrada

Haverão múltiplos casos de teste. Cada caso inicia com uma linha contendo n indicando o número de nós, numerados de 1 até n (um valor $n = 0$ indica o fim das entradas). A próxima linha irá conter um inteiro e indicando o número de arestas. Iniciando na linha seguinte, estarão os pares $n_1 n_2$ indicam uma aresta não direcionada entre os nós n_1 e n_2. Esta informação poderá se estender por várias linhas. Você pode assumir que $n \leq 100$ e que $e \leq 300$.

Saída

Para cada caso de teste gerar uma linha de saída com a mensagem `sim` ou `nao`.

Exemplo de Entrada

```
22
21
1 2 2 3 2 4 2 5 2 6 6 7 6 10 10 8 9 10 10 12 11 12 12 13 12 17
18 17 15 17 15 14 16 15 17 20 20 21 20 22 20 19
16
15
1 2 2 3 5 2 4 2 2 6 6 7 6 8 6 9 9 10 10 12 10 11 10 14 10 13 13 16 13 15
0
```

Exemplo de Saída

```
nao
sim
```

3 Problema C: Pedras Coloridas

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

Descrição

Os alunos do DCC estão constantemente reunidos no *hall* do bloco F a espera das aulas, jogando, lendo, ou desafiando alguém com algum quebra-cabeça.

Alguém veio com o jogo das predras coloridas. Neste jogo, é dada uma linha de m pedras coloridas, num total de k cores diferentes. Qual é o número mínimo de pedras que devem ser removidas de modo que não existam duas pedras de mesma cor separadas nesta linha?

Entrada

A entrada contém vários cenários. Cada cenário começa com uma linha contendo os inteiros m e k, onde $1 < m < 100$ e $1 < k < 5$. A próxima linha contém m inteiros $x_1, x_2, \ldots, x_m$ cada qual tem um valor do conjunto $\{1, 2, \ldots, k\}$, onde k representa as k diferentes cores das pedras. O final de arquivo é marcado com $m = k = 0$ e esta entrada não deve ser processada.

Saída

Para cada caso de teste, o programa deverá imprimir o número mínimo de pedras a serem removidas para que a condição seja satisfeira.

Exemplo de Entrada

```
10 3
2 1 2 2 1 1 3 1 3 3
0 0
```

Exemplo de Saída

```
2
```

No exemplo acima, a solução ótima foi alcançada pela remoção da 2ª e 7ª pedra, restando três pedras da cor `2`, três pedras da cor `1`, e duas pedras da cor `3`. Outras soluções são possíveis, mas a saída `2` é o menor valor de remoções.

4 Problema D: Polyana Okimoto

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

Descrição

Polyana é uma nadadora da modalidade "maratona em águas abertas". Sim, ela conseguiu antecipadamente uma vaga para Londres devido seu desempenho nestes últimos anos na prova de 10km. Como voce ela é uma maratonista, sabe que nada ocorre por acaso, mas sim com muito treino.

Nesta fase final de preparação para as olímpiadas, ela precisa aumentar a sua velocidade e para isto ela precisa controlar a sua passagem a cada 100m. Assim, ela precisa saber o menor e o maior tempo de passagem a cada 100m, durante os seus treinos de média distância. Ajude o técnico de Polyana a dizer a ela quando foi o melhor e o pior tempo nas passagens de 100m.

Entrada

O arquivo de entrada contém vários cenários. Cada entrada de teste começa com um valor m, tal que $1 < m < 50$. A próxima linha contém o tempo tomado nas passadas de 100m que seu técnico registrou no cronômetro. O final de arquivo é marcado com $m = 0$, sendo que esta entrada não deve ser processada.

Entrada

O formato da entrada é dado por:

```
3
01min05s 02min15s 03min25s
0
```

Assim, ela nadou um total de 300m, sendo que o tempo necessário para cada passada de 100m foi de: 1min05s, 1min10s e 1min10s, respectivamente, nos primeiros 100m, na passagem dos 200m e nos últimos 100m. O valor `min` refere-se a minutos, e s aos segundos, ambos no intervalo $[00, 59]$. Esta sessão de treino sempre dura menos do que 1 hora.

Saída

Para cada caso de teste, o programa deverá imprimir em uma linha o melhor e o pior tempo entre as passagens de 100m, no formato abaixo:

```
01min05s 01min10s
```

Como curiosidade, isto é algo normal para um nadador olímpico, pois ele conta o número de braçadas e sabe com uma certa precisão o seu tempo a cada 100m.

Exemplo de Entrada

```
3
01min05s 02min15s 03min25s
4
01min05s 02min04s 03min17s 04min25s
0
```

Exemplo de Saída

```
01min05s 01min10s
00min59s 01min13s
```

5 Problema E: *Scrabble*

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

O Problema

Scrabble é jogado com peças que podem ter ou não uma letra impressa. No útimo caso a peça pode ser usada para representar uma letra de sua escolha. Na sua vez, você disporá as peças de modo a formarem uma palavra. Cada peça pode ser usada somente uma vez, mas nem todas as peças precisam ser usadas. Dadas várias peças de *Scrabble* e um dicionário, determine quantas palavras podem ser formadas usando as peças dadas de *Scrabble*.

Entrada

O arquivo de entrada conterá múltiplos casos-teste. Em cada caso, a primeira linha contém um inteiro positivo $n \leq 1000$ indicando o número de palavras no dicionário. A seguintes n linhas contêm, cada uma, uma única *string* (palavra) contendo entre 1 e 7 letras maiúsculas, representando uma palavra no dicionário. Nenhuma palavra aparecerá no dicionário duas vezes. A linha seguinte conterá uma única *string* informando as peças que você tem disponíveis. Ela conterá somente letras maiúsculas, representando peças com letras nelas, e *underscores*, representando peças em branco. A *string* conterá de 1 a 7 caracteres, possivelmente incluindo peças duplicadas. O final do arquivo de entrada (*end-of-file*) é marcado por um caso-teste, com $n = 0$ e não deve ser processado.

Saída

Para cada caso-teste, escreva uma única linha com o número de palavras do dicionário que que devem ser soletradas com as dadas peças de *Scrabble*.

Exemplo de Entrada

```
5
PROGRAM
CONTEST
PIZZA
ZA
PITA
_PIZA
3
BANANAS
CARROTS
FIGS
A__AA__
0
```

Exemplo de Saída

```
3
2
```

No primero case-teste, `PIZZA`, `ZA` e `PITA` podem ser soletrados como

`PIZ_A, ZA e PI_A`

Não há letras suficientes para soletrar `PROGRAM` ou `CONTEST`. No segundo caso-teste, `BANANAS` e `FIGS` podem ser soletrados como

`_A_A_A_ e ____`

Por outro lado, `CARROTS` iria requerer 6 espaços, adicionalmente ao A.

6 Problema F: A Pontuação do Campeonato Mudou!

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

Descrição

Se já não bastasse o sufoco e a pressão que o técnico Bernardinho do volei brasileiro tem passado, as regras da classificação nas olimpíadas mudaram! Além da vitória, agora o número de *sets* vencidos também conta!

A regra é a seguinte: cada jogo é composto por no máximo 5 *sets* num esquema melhor-de-três; o primeio time que vencer 3 *sets* vence o jogo, sendo que a pontuação final é a diferença entre os *sets* ganhos e os perdidos para o time vencedor e 0 para o time perdedor.

Ajude o técnico Bernadinho a computar os resultados dos jogos e dizer a ele a ordem de classificação de cada equipe.

Entrada

A entrada do arquivo de teste tem vários cenários. Em cada linha consta o número m indicando quantos times estão disputando a olimpíada. Nas m linhas seguintes, constam os resultados de cada partida entre os m times, em ordem, de 1 a m (a célula m_{ij} corresponde ao resultado do jogo do time i contra o time j). O número de times m é dado por $1 \leq m \leq 16$. O final de arquivo é marcado com $m = 0$ e esta entrada não será processada.

O formato da entrada pode ser visto em *Exemplo de Entrada*, abaixo. Neste exemplo, a linha após $m = 4$ corresponde aos resultados do time 1, até os resultados do time 4. O resultado `0x0` significa que o time não disputa contra si mesmo. Os resultados da tabela são simétricos, isto é, se o time i ganhou do time j por `3x2`, na linha de j, tal resultado vai aparecer como `2x3`.

Saída

Para cada caso de teste, o programa deverá imprimir a ordem de classificação dos times, do melhor colocado até o pior. Em caso de empate, o time com o menor índice ganha a disputa de empate. Para o exemplo da seção seguinte, tem-se uma saída igual a:

```
3 1 2 4
```

O time `3` foi o vencedor com `4` pontos `(3x0=3 + 3x2=1+ 0x0=0 + 2x3=0)=4`, e o time `4` o último colocado com `1` ponto `(1x3=0 + 1x3=0 + 3x2=1 + 0x0=0)=1`.

Exemplo de Entrada

```
4
0x0 3x2 0x3 3x1
2x3 0x0 2x3 3x1
3x0 3x2 0x0 2x3
1x3 1x3 3x2 0x0
0
```

Exemplo de Saída

```
3 1 2 4
```

7 Problema G: Escrevendo Números

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

Seu amigo Arthur Dent lhe trouxe um presente do planeta *Betelgeuse*. O presente é um manuscrito (veja a tabela de codificação: tabela 1) que relaciona um número com a sua escrita por extenso. Arthur se entreteve bastante com o manuscrito até que decidiu que aquilo definitivamente não era legal e resolveu dar de presente para você e aproveitar o resto do tempo tomando algumas *Dinamites Pangaláticas*.

Sabendo que você é um excelente programador, ele lhe desafiou a decifrar a lógica–numérica deste manuscrito. Assim, solicitou que você escrevesse um programa que dado qualquer número **n**, este retornasse este número por extenso de acordo com a tabela 1. Arthur disse que se você conseguir, você irá ganhar um balão *Vogon*[1].

Tabela 1: Tabela de Conversão do planeta *Betelgeuse*

1	u	2	d	3	t
4	q	5	c	6	s
7	e	8	o	9	n
10	uA	11	uAu	12	uAd
20	dA	30	tA	40	qA
42	qAd	50	cA	60	sA
70	eA	80	oA	90	nA
99	nAn	100	uB	101	uBu
900	nB	999	nBnAn	1000	uAB
9900	nAnB	10000	uC	99999	nCnAnBnAn
123456	uAdCtAqBcAs	1234567	uBdAtCqAcBsAe	12345678	uAdBtAqCcAsBeAo
100000000	uD	10^{16}	uE	10^{32}	uF

Como Arthur está esgotado de carregar o tal balão *Vogon* de lá para cá ele lhe deu algumas dicas para decifrar este manuscrito, cuja base é a tabela 1:

- As letras minúsculas representam os dígitos de 1 até 9, todos são mostrados no manuscrito;
- As letras maiúsculas são fatores e sua ordem é crescente, dada por: `A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z`;
- Observe como todos os fatores menores são utilizados na frente de um fator maior:
  - $uB = 100$: 1×100;
  - $nB = 900$: 9×100;
  - $nAnB = 9900$: $nAn = 99$, 99×100;
  - $nAnBnC = 99090000$: $nAnBn = 9909$, 9909×10000;
- Não entre em pânico!

[1] Mas, terá que buscar este balão lá em *Betelgeuse*.

Entrada

A entrada consiste de vários casos de teste. Cada caso de teste é uma linha contendo um número n em formato decimal, tal que $0 < n \leq 10^{512}$. A entrada termina com $n = 0$.

Saída

Para cada caso de teste seu programa deve imprimir uma linha contendo o número no formato do manuscrito.

Exemplo de Entrada

```
42
9909
0
```

Exemplo de Saída

```
qAd
nAnBn
```

8 Problema H: The Settlers of Catan

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

Description

Within Settlers of Catan, the 1995 German game of the year, players attempt to dominate an island by building roads, settlements and cities across its uncharted wilderness.

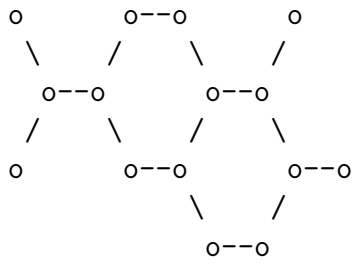
You are employed by a software company that just has decided to develop a computer version of this game, and you are chosen to implement one of the game's special rules: When the game ends, the player who built the longest road gains two extra victory points.

The problem here is that the players usually build complex road networks and not just one linear path. Therefore, determining the longest road is not trivial (although human players usually see it immediately).

Compared to the original game, we will solve a simplified problem here: You are given a set of nodes (cities) and a set of edges (road segments) of length `1` connecting the nodes.

The longest road is defined as the longest path within the network that doesn't use an edge twice. Nodes may be visited more than once, though.

Example: The following network contains a road of length `12`.



Input

The input file will contain one or more test cases.

The first line of each test case contains two integers: the number of nodes n ($2 \leq n \leq 25$) and the number of edges m ($1 \leq m \leq 25$). The next m lines describe the m edges. Each edge is given by the numbers of the two nodes connected by it. Nodes are numbered from 0 to $n - 1$. Edges are undirected. Nodes have degrees of three or less. The network is not neccessarily connected.

Input will be terminated by two values of 0 for n and m.

Output

For each test case, print the length of the longest road on a single line.

Sample Input

```
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
```

Sample Output

```
2
12
```

9 Problema I: Dropping Tests

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

Description

In a certain course, you take n tests. If you get a_i out of b_i questions correct on test i, your cumulative average is defined to be:

$$\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}$$

Given your test scores and a positive integer k, determine how high you can make your cumulative average if you are allowed to drop any k of your test scores.

Example

Suppose you take 3 tests with scores of 5/5, 0/1, and 2/6. Without dropping any tests, your cumulative average is $100.\frac{5+0+2}{5+1+6} = 50$. However, if you drop the third test, your cumulative average becomes $100.\frac{5+0}{5+1} \approx 83.33 \approx 83$.

Input

The input test file will contain multiple test cases, each containing exactly three lines. The first line contains two integers, $1 \leq n \leq 1000$ and $0 \leq k < n$. The second line contains n integers indicating a_i for all i. The third line contains n positive integers indicating b_i for all i. It is guaranteed that $0 \leq a_i \leq b_i \leq 1,000,000,000$ The end-of-file is marked by a test case with $n = k = 0$ and should not be processed. For example:

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

Output

For each test case, write a single line with the highest cumulative average possible after dropping k of the given test scores. The average should be rounded to the nearest integer. For example:

```
83
100
```

To avoid ambiguities due to rounding errors, the judge tests have been constructed so that all answers are at least 0.001 away from a decision boundary (i.e., you can assume that the average is never 83.4997).

10 Problema J: Sequências

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

Descrição

Dorotéia, prima de Micalatéia, observou um fato interessante: uma sequência de números pode ser formada pela adição do quadrado de cada dígito ($452 = 4^2 + 5^2 + 2^2 = 45$) de um número para formar um novo número até que esta caia em um ciclo.

Por exemplo, partindo de 44 temos: $44 \rightarrow 32 \rightarrow 13 \rightarrow 10 \rightarrow 1 \rightarrow 1$, enquanto que, partindo de 85 temos a sequência: $85 \rightarrow 89 \rightarrow 145 \rightarrow 42 \rightarrow 20 \rightarrow 4 \rightarrow 16 \rightarrow 37 \rightarrow 58 \rightarrow 89$ (percebe-se uma repetição nestes dois casos: a primeira sequência convergirá para o valor 1 e a segunda sequência ira sempre repetir o ciclo iniciado pelo número 89).

Dorotéia, fascinada pelas sequências, pediu ajuda à Micalatéia, mas como ela estava muito ocupada com outras tarefas, decidiu pedir a ajuda aos maratonistas para desvendar os mistérios destas sequências. Dorotéia irá fornecer uma faixa de valores naturais [a,b] para números iniciais da sequência, para que você determine quantas sequências convergirão para o valor 1 e quantas ficarão presas em um ciclo iniciado pelo valor 89.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste é composto por dois inteiros a e b, onde $0 \leq a < b \leq 10^7$. O final da entrada é marcado por $a = 0$ e $b = 0$ (zeros são utilizados somente para marcar o final da entrada).

Saída

Para cada caso de teste seu programa deve imprimir uma linha dois inteiros separados por um espaço, o primeiro sendo o número de sequências presas no valor 1 e o segundo o número de sequências presas no valor 89.

Exemplo de Entrada

```
8 9
5 102
0 0
```

Exemplo de Saída

```
0 2
19 79
```