3ª Maratona Doméstica de Programação da UDESC

Prova Principal

Joinville, 23 de agosto de 2013

Sevidor BOCA:

`http://10.20.107.207/`
(acesso interno)

`http://200.19.107.207/`
(acesso externo)

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Lucas Hermann Negri (coordenação técnica), Yuri Kaszubowski Lopes, Rafael Parpinelli, Adriano Fiorese, Alexandre Gonçalves Silva, Roberto Silvio Ubertino Rosso Jr., Rogério Eduardo da Silva

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 também está disponível nas máquinas Linux, que pode ser utilizada em vez da Unity. Para isto, basta dar *logout*, e selecionar a interface KDE. Usuário e senha: *udesc*

Patrocinador e Agradecimentos

- Linx – Patrocinador oficial do ano de 2013;
- Realização: DCC/UDESC;
- Rutes, pelo empenho nos treinos e no projeto;
- Alguns, muitos outros anônimos.

Conteúdo

1 Problema A: Anéis e Runas

`Arquivo: aneis.[c|cpp|java]` `Tempo limite: 10 s`

Frodo e seus amigos entraram nas minas de Moria se depararam com uma série de portões. Em cada portão está escrita uma charada descrevendo o estado de um conjunto de anéis que controlam aquele portão em particular. Examinando a charada, Frodo pode determinar se o portão pode ou não ser aberto.

Uma **charada** consiste de múltiplas **runas**. Uma **runa** válida contém exatamente 3 **afirmações** sobre 3 **anéis**. Cada **afirmação** em uma **runa** é **verdadeira** ou **falsa**, dependendo do **estado** (girando ou parado) de um anel específico.

Para abrir os portões, os hobbits precisam ler a charada e, então, decidir quais dos anéis devem girar, e quais devem ficar parados. Quando os anéis corretos estiverem girando, eles recitam um encantamento, e, se a charada para o portão for satisfeita, o portão abrirá. Para a charada ser satisfeita, cada runa dela deve ter **ao menos** uma afirmação na qual é verdade.

Por exemplo, considera a seguinte runa: `1 -2 3 0`. Esta runa será verdadeira se: (o anel 1 estiver girando) **OU** (o anel 2 não estiver girando) **OU** (o anel 3 estiver girando). O `0` indica o final da runa. Aqui, se o número do anel for positivo, ele deve girar para satisfazer a runa; se for negativo, ele deve ficar parado.

Um mesmo anel só pode aparecer uma vez em uma dada runa, mas pode ser utilizado mais de uma vez na charada inteira.

Entrada

A entrada é descrita a seguir:

- A primeira linha contém um único inteiro G ($1 \leq G \leq 30$), que denota o número de portões com charadas a serem resolvidas. Cada portão é um caso de teste individual.
- A primeira linha para cada portão contém dois inteiros, *aneis* ($3 \leq aneis \leq 22$) e *runas* ($1 \leq runas \leq 100$), separados por um espaço. *aneis* é o número de anéis disponíveis, sendo que cada anel é numerado de 1 até *aneis*; *runas* é o número de runas que precisam ser satisfeitas para o portão se abrir.
- As próximas *runas* linhas descrevem as runas individualmente, que especificam as relações entre os anéis para aquele portão. Cada runa é representada por uma única linha contendo quatro números: r_1, r_2, r_3, e 0, separados por espaços, que relacionam três anéis com a runa em questão.

Saída

O seu algoritmo deve imprimir uma linha para cada portão. Os seguintes casos devem ser verificados, sendo que eles estão listados em ordem de prioridade, isto é, o primeiro que for verdade deve ser considerado, ignorando os outros:

- Se qualquer runa em uma charada contiver uma afirmação sobre um anel nulo (igual a 0 ou -0), a charada será inválida. Imprima:
  "`INVALIDO: ANEL NULO`".
- Se qualquer runa em uma charada contiver uma afirmação referenciando um anel inexistente, imprima: "`INVALIDO: ANEL FALTANTE`".

- Se qualquer runa se referir a um anel mais do que uma vez, imprima: "`INVALIDO: ANEL REPETIDO`".
- Se houver uma configuração de anéis que satisfizer todas as runas na charada, imprima: "`RUNAS SATISFEITAS`"
- Se não houver uma configuração válida de anéis, imprima: "`CHARADA IMPOSSIVEL`"

Exemplo de Entrada	**Exemplo de Saída**
5 3 5 1 2 3 0 1 -2 3 0 1 3 -2 0 -3 -1 2 0 1 2 3 0 3 8 3 1 2 0 3 -1 2 0 3 1 -2 0 3 -1 -2 0 2 1 -3 0 -2 1 -3 0 -1 2 -3 0 -1 -2 -3 0 3 2 -1 1 3 0 0 1 3 0 3 2 -1 1 3 0 7 1 3 0 3 2 -1 1 3 0 2 1 3 0	RUNAS SATISFEITAS CHARADA IMPOSSIVEL INVALIDO: ANEL NULO INVALIDO: ANEL FALTANTE INVALIDO: ANEL REPETIDO

2 Problema B: Batalha na Terra-Média

`Arquivo: batalha.[c|cpp|java]` `Tempo limite: 2 s`

Arda está novamente em guerra. Por enquanto, os confrontos entre os exércitos do bem e os exércitos do mal estão restritos a Terra-Média, mas contam com a participação de diferentes raças. Cada raça possui um determinado *valor* em batalha, como visto no quadro abaixo:

Lado do Bem	**Lado do Mal**
Hobbits - 1	Orcs - 1
Homens - 2	Homens - 2
Elfos - 3	Wargs - 2
Anões - 3	Goblins - 2
Águias - 4	Uruk Hai - 3
Magos - 10	Trolls - 5
	Magos - 10

Apesar do local, do clima e de outros fatores serem importantes na batalha, você deverá comparar o exército do bem contra o exército de Sauron utilizando simplesmente o *valor* em batalha total de cada exército. Com base na contagem de cada raça do lado do bem, e na contagem de cada raça do lado do mal, determine qual lado vencerá.

Entrada

A primeira linha da entrada conterá o número de batalhas a serem processadas. Cada batalha será descrita por duas linhas. A primeira linha é composta pela contagem de cada raça do lado do bem, na ordem: Hobbits, Homens, Elfos, Anões, Águias e Magos. A segunda linha é composta pela contagem das raças do lado do mal, na ordem: Orcs, Homens, Wargs, Goblins, Uruk Hai, Trolls e Magos. Todos os números são inteiros não-negativos, sendo que cada número e o valor total de cada lado podem ser representados em 32 bits.

Saída

Para cada batalha imprima uma linha contendo **Batalha #: O Bem triunfou contra o Mal** se o lado do bem venceu, **Batalha #: O Mal erradicou o Bem** se o lado do mal venceu ou **Batalha #: Nenhum vencedor no campo de batalha** em caso de empate, onde # deve ser substituído pelo número da batalha atual.

Exemplo de Entrada	**Exemplo de Saída**
3 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 10 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0	Batalha 1: O Mal erradicou o Bem Batalha 2: O Bem triunfou contra o Mal Batalha 3: Nenhum vencedor no campo de batalha

3 Problema C: Construindo a Torre

`Arquivo: construindo.[c|cpp|java]` `Tempo limite: 2 s`

Saruman ordenou que o seu exército de orcs extraia, por N dias consecutivos, minério e madeira da terra no entorno da grande torre de Orthanc. No i-ésimo dia (contado do dia 0 até o dia N, inclusive), Saruman decide se gasta os recursos em novos equipamentos de guerra para o seu exército ou se gasta com a adição de novos andares na sua torre.

Como estratégia para gerenciar os recursos, Saruman decidiu adicionar um andar à torre sempre que a representação em binário do dia atual tiver um número total de 1**'s múltiplo de** 3. Por exemplo, Saruman adicionará um andar à torre no dia 7 ($7_{10} = 111_2$), no dia 11 ($11_{10} = 1011_2$), e então nos dias 13, 14, 19 e assim por diante.

Saruman está muito ocupado com toda esta preparação para batalha, e pediu a você para prever quantos andares serão adicionados à torre após N dias. Você pode escrever um programa para auxiliá-lo?

Entrada

O arquivo de entrada contém múltiplos casos de teste, cada um em uma linha. Cada caso de teste é composto por um inteiro N ($0 \leq N \leq 10^{16}$). A entrada termina com um final de arquivo (isto é: **eof**).

Saída

Para cada caso de teste imprima uma linha contendo:

`Dia N: Andares = L`

onde `N` é o dia fornecido na entrada e `L` é a quantidade de andares adicionados até o dia `N`, inclusive.

Exemplo de Entrada	**Exemplo de Saída**
2 19 64	Dia 2: Andares = 0 Dia 19: Andares = 5 Dia 64: Andares = 21

4 Problema E: Energia do Feitiço

`Arquivo: energia.[c|cpp|java]` `Tempo limite: 2 s`

O lançamento de um feitiço é um processo contínuo, que inicia com uma certa quantidade de energia (medida em *mana*) que pode ser usada para invocar elementos para o feitiço. Cada elemento pode ser invocado instantaneamente, em qualquer quantidade (até frações), pelo consumo de *mana*. Após invocado, o elemento se torna (por toda a duração do feitiço) uma fonte de *mana*, com uma determinada potência (medida em *mana* por segundo). Estas fontes de *mana*, como esperado, fornecem *mana* continuamente, que pode por sua vez ser utilizada para invocar novos elementos. Este processo é continuado até que a potência total (o somatório das potências de cada elemento invocado) alcance um determinado limiar, que permite o lançamento do feitiço.

Existe um detalhe neste processo, na qual feiticeiros experientes irão explorar para lançar feitiços mais eficientemente. Cada elemento pode ter até um elemento pai para dar suporte a sua invocação, reduzindo o custo deste elemento pela metade, caso o elemento pai já esteja presente no feitiço. Por exemplo, se o elemento A dá suporte simultaneamente ao elemento B e ao elemento C, pode-se invocar 5 unidades do elemento A com o custo total e então invocar 3 unidades do elemento B e 2 unidades do elemento C com o custo de *mana* reduzido pela metade. Se quisermos então invocar mais 1 unidade do elemento C, esta porção estará com o custo total novamente, uma vez que todas as unidades do elemento A já estão dando suporte a outros elementos. Note que todos os 3 elementos contribuem com suas potências completas para o feitiço; dar suporte não interfere na potência de saída e nem consome um elemento.

Dada uma quantidade inicial de *mana*, uma quantidade alvo de potência e uma descrição dos elementos do feitiço disponíveis para invocar, encontre como lançar um feitiço alcançando a quantidade desejada de potência em uma quantidade mínima de tempo.

Entrada

As entradas são constituídas por múltiplos casos de teste. Cada caso de teste se inicia com uma linha com 3 inteiros separados por espaço. N, E, e P, denotam o número de elementos, a *mana* inicial, e a potência alvo, respectivamente. Seguindo esta linha estão N outras linhas, a $i^{\text{ésima}}$ linha descreve o elemento i por três inteiros separados por espaço, e_i, p_i, e pai_i. Estes denotam o custo de *mana* para invocar uma unidade este elemento, a potência por unidade e o índice do elemento pai (1-indexado; $pai_i = 0$ se o elemento i não tem elemento pai), respectivamente.

Restrições:

- $1 \leq N \leq 1000$, $1 \leq E \leq 10^9$, $1 \leq P \leq 10^9$.
- $1 \leq e_i \leq 10^9$ para todo i, $0 \leq p_i \leq 10^9$ para todo i, $0 \leq pai_i \leq N$ para todo i.
- Pelo menos um p_i deve ser positivo.
- Nenhum elemento é o seu próprio antecessor; em outras palavras, nenhum elemento é capaz de dar suporte a si próprio, seja direta ou indiretamente.

As entradas são terminadas no caso onde $N = E = P = 0$.

Saída

Para cada caso de teste, imprima uma única linha com um único inteiro que represente o número mínimo de segundos necessários para alcançar a potência desejada (arredondando para cima para o inteiro mais próximo).

Exemplo de Entrada	**Exemplo de Saída**
1 1 3 2 1 0 1 1 1000000 200 100 0 2 1 1000000 200 100 0 2 1 1 2 1 1000000 200 100 2 2 1 0 0 0 0	4 30 29 14

5 Problema F: Ferreiro Élfico

`Arquivo: ferreiro.[c|cpp|java]` `Tempo limite: 2 s`

Nas raças élficas da Terra-Média acreditava-se que certos números eram mais significativos do que outros. Quando se usa uma determinada quantidade n de metal para forjar uma espada em particular, eles acreditavam que esta espada seria mais poderosa se a sua espessura k fosse escolhida de acordo com a seguinte regra:

Dado um número inteiro não-negativo n, qual é o menor valor de k tal que as representações decimais de inteiros na sequência:

n	2n	3n	4n	5n	...	kn

tal que concatenados contenham todos os dez dígitos (de 0 até 9) pelo menos uma vez?

O Senhor Elrond de Rivendell encomendou-lhe uma tarefa de desenvolver um algoritmo que encontre a espessura ótima (k) para qualquer quantidade de metal dada (n).

Entrada

Cada caso de teste consistirá de uma linha contendo um inteiro n. O final das entrada é sinalizado por um fim de arquivo. O inteiro está na faixa de `1` a `200000000`, inclusive.

Saída

Para cada caso de teste, imprima um inteiro em uma linha, indicando o valor de k necessário tal que cada dígito de 0 até 9 estejam presentes pelo menos uma vez na sequência.

Exemplo de Entrada	**Exemplo de Saída**
1 10 123456789 3141592	10 9 3 5

Exemplo

Quando $n = 123456789$, precisamos de $k >= 3$ para encontrarmos todos os 10 dígitos: $digitos = 123456789, 246913578, 370370367$ (note que o dígito 0 só aparece no terceiro termos, enquanto que todos os outros já apareceram no primeiro termo). A **‘,’** (vírgula) no exemplo separa cada uma das sequências kn.

6 Problema G: Gandalf

`Arquivo: gandalf.[c|cpp|java]` `Tempo limite: 2 s`

Os textos de Gandalf há muito tempo estão disponíveis para estudo. Porém até agora ninguém havia descoberto em que linguagem estão escritos. Recentemente, devido ao trabalho de programação feito por um hacker conhecido apenas pelo codinome de PODRE13, foi descoberto que Gandalf utilizava apenas um esquema simples de substituição de letras. E ainda mais, que este é sua própria inversa, ou seja, a mesma operação que embaralha a mensagem a desembaralha.

Esta operação é executada através da substituição de vogais na sequência:

(a i y e o u)

com a vogal avançando **três posições** ciclicamente, preservando caso (ou seja, minúscula ou maiúscula). De forma semelhante, consoantes são substituídas a partir da sequência:

(b k x z n h d c w g p v j q t s r l m f)

Agora avançando **dez letras**. Assim, por exemplo, na frase:

Um anel para governar a todos.

Traduz-se para:

Yw etac bede fikadted e nirih.

O fascinante sobre essa transformação, é que a linguagem resultante ainda produz palavras pronunciáveis. Para este problema, você vai escrever um código que traduza os manuscritos de Gandalf em um texto simples (com letras sem acentos nem cedilhas).

Entrada

O arquivo de entrada irá conter vários casos de teste. Cada caso de teste é composto por uma única linha contendo até 100 caracteres, que representa um texto escrito por Gandalf. Todos os carácteres são ASCII, no intervalo do **"espaço"** (32) até o **"til"** (126), mais uma nova linha terminando cada linha. O final da entrada é indicado com o **"fim de arquivo"** (EOF).

Saída

Para cada caso de teste de entrada, imprima sua tradução em texto simples (sem acentos e sem formatação). A saída deve conter exatamente o mesmo número de linhas e caracteres que a entrada.

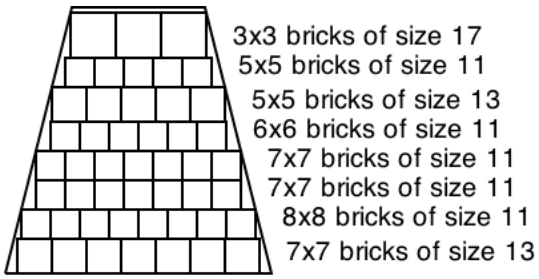
Exemplo de Entrada	**Exemplo de Saída**
`Ita dotf ni dyca nsaw ecc.` `Um anel para governar a todos.`	`One ring to rule them all.` `Yw etac bede fikadted e nirih.`

7 Problema H: Honra dos Anões

`Arquivo: honra.[c|cpp|java]` `Tempo limite: 20 s`

Os anões da Terra Média são reconhecidos por suas habilidades de ferreiros e mineiros, mas eles são também mestres construtores. Durante a época dos dragões, os anões descobriram que os edifícios mais resistentes a ataques eram as pirâmides quadradas truncadas (uma pirâmide que não sobe todas as paredes até o fim, mas que no topo resta um quadrado).

Os anões sabiam que a forma ideal dessa construção baseava-se na altura que eles quisessem que ela tivesse, e no tamanho do quadrado da base e do topo. Eles tinham, tipicamente, três (tipos) tamanhos de tijolos cúbicos com os quais trabalhavam. A meta deles era maximizar o volume de tal construção baseada nas seguintes regras:



Height:100; bottom: 100; top: 50

A construção é erguida camada à camada. Cada camada é composta de um quadrado de tijolos de um único tamanho. Nenhuma parte de tijolo pode ultrapassar a forma ideal, seja nos lados ou no topo. A estrutura resultante terá lados com reentrâncias (denteados) e pode ser mais curta que o formato ideal, porém deve caber completamente dentro do projeto ideal. A Figura à direita mostra uma seção vertical de uma dessas pirâmides.

Não há limite no número de tijolos de cada tipo que podem ser utilizados.

Entrada

Cada linha da entrada de dados deverá conter 6 valores, cada valor é separado por um espaço simples. Os três primeiros valores representam a altura ideal do templo, o lado do quadrado da base da construção, o lado do quadrado no topo da construção (todos os três valores inteiros não negativos menores ou iguais a um milhão). Os próximos três valores da entrada representam os lados dos três tipos de tijolos (todos os três valores inteiros não negativos menores ou iguais a dez mil). A entrada de dados é finalizada com um final de arquivo.

Saída

Para cada linha de entrada, mostre o volume máximo possível com base nas regras já mencionadas.

Exemplo de Entrada	**Exemplo de Saída**
`500000 800000 300000 6931 11315 5000`	`160293750000000000`

8 Problema I: Imagine um Cubo

`Arquivo: imagine.[c|cpp|java]` `Tempo limite: 100 s`

Uma vez descoberto que o anel do poder estava em posse de Frodo, Gandalf imediatamente saiu em busca do conselho de Saruman, o chefe de sua ordem. Saruman discordou da opinião de Gandalf, de que eles deveriam destruir o anel. Ele decidiu que não poderia permitir imediatamente que Gandalf saisse livre e fosse ajudar Frodo. Assim ele trancou Gandalf no quarto mais alto da sua torre escura em Isengard. Para manter Gandalf ocupado, ele colocou uma única porta no quarto, do qual apenas se poderia escapar solucionando um enigma.

Depois de rapidamente procurar por uma saída do quarto, Gandalf percebeu uma fechadura muito peculiar numa porta que parecia levar ao telhado da torre. A fechadura era um cubo básico de seis lados sem marcas em quaisquer de suas superfícies ou lados. Este cubo estava sobre uma área retangular, uma *grid* de tamanho $m \times n$. A grid tinha exatamente seis quadrados pintados. Gandalf percebeu que quando ele rolava uma face sem cor do cubo sobre um quadrado que tinha marcas pintadas nele, as marcas eram transferidas da grid para a face do cubo.

Quando o cubo era rolado sobre um quadrado **sem marcas** pintadas nele e a superfície de contato do cubo tinha uma marca nela, a marca era transferida para a *grid*. Se o cubo fosse rolado sobre um quadrado da *grid* que tivesse uma marca e a superfície de contato do cubo tivesse uma marca de qualquer tipo nela, então nada acontecia.

Gandalf supôs que somente poderia abrir a porta que levava para o telhado se o cubo fosse rolado numa sequência de movimentos em direção do quadrado alvo, tal que todas as marcas da *grid* fossem transferidas para o cubo. Além disto, para escapar da armadilha de Saruman, ele supôs (corretamente) que precisaria executar a tarefa com um número mínimo de movimentos.

Dada uma configuração inicial de pintura nos quadrados (leia-se na *grid*), uma localização inicial do cubo, uma localização alvo desejada, qual é o numero mínimo de movimentos que Gandalf deve realizar para conseguir transformar o cubo na configuração desejada, ou seja, com pintura em todos os seus lados?

Para o primeiro exemplo (veja abaixo), uma amostra de solução seria: para baixo, direita, direita, para cima, direita, direita, para baixo, esquerda, direita, esquerda. Isto é, 10 movimentos.

Entrada

O arquivo de entrada contém múltiplos casos teste. Cada caso teste consistirá de uma *grid*, área retangular, com $m \times n$ caracteres, onde m e n estão cada um entre os valores `2` e `20`. Dentro de cada caso teste, espaços vazios serão descritos por '`.`', quadrados pintados por '`P`', quadrados ilegais por '`#`', a posição inicial do cubo por '`C`' (***C**ube*), e o quadrado alvo por '`G`' (***G**oal*, alvo em inglês). Para cada caso de teste haverá exatamente um '`C`', um '`G`', 6 '`P`'s e não mais do que 12 '`.`'s. Os casos de teste de entrada serão separados por uma única linha branca. A entrada será concluída pelo final do arquivo (**eof**).

Saída

Para cada caso teste de entrada, imprima o número mínimo de movimentos necessários para obter a configuação desejada. Se não for possível obter esta configuração, imprima -1.

Exemplo de Entrada	**Exemplo de Saída**
C.PPP PPPG. C.... G##.P .##PP ..PPP PPPPPP.... #########. #########. #########. #########. #########. #########. #########. #########. #########C #########G PPP PCP PG. .PPPCPPP.. ....G..... 	10 23 -1 15 21

9 Problema J: Jogo do WIN

`Arquivo: jogo.[c|cpp|java]` `Tempo limite: 60 s`

Quando Frodo, Sam, Merry e Pippin estão no Pônei Saltitante bebendo cerveja, eles gostam de brincar de um jogo com pergaminho e caneta para decidir quem vai pagar a próxima rodada de cerveja. O jogo funciona da seguinte forma:

Dado uma área retangular com ladrilhos dentro dela, uma *grid* (matriz), com um total $m \times n$ ladrilhos nesta área, onde cada um destes ladrilhos está marcado com uma magia sobre eles: `W`, `I`, e `N`.

O objetivo é encontrar o número trinômios maximais que podem ser cortados destes ladrilhos, tal que cada trinômio seja formado pela palavra `WIN` (do inglês: **vencer**). Naturalmente, os trinômios possíveis são somente aqueles com três azulejos consecutivos em linha reta e aqueles em formato de '**L**'. O Hobbit que tiver a habilidade de encontrar o maior número destes maximais vence e escolhe quem vai pagar a próxima rodada de cerveja. Sua tarefa é encontrar o número de maximais dentro desta *grid* retangular de ladrilhos.

Um detalhe: Sam e Pippin tendem a pagar a maioria das rodadas de cerveja quando jogam este jogo, então eles estão fazendo *lobby* para mudarem de jogo para um novo jogo: Rocha, Pergaminho, Espada (RPS)!

Entrada

O arquivo de entrada contém múltiplos casos de teste. Cada caso de teste consiste de uma matriz (*grid*) retangular de tamanho $m \times n$ (onde $1 \leq m, n \leq 30$) contendo somente letras `W`, `I`, e `N`. Cada caso de teste são separados por uma linha em branco. As entradas serão terminadas por um fim de arquivo.

Saída

Para cada caso de teste, imprima uma linha contento um único inteiro indicando o número total máximo de ladrilhos que possam ser formados.

Exemplo de Entrada	**Exemplo de Saída**

`WIIW`
`NNNN`
`IINN`
`WWWI`
 	
`NINWN`
`INIWI`
`WWWIW`
`NNNNN`
`IWINN`
	`5`
	`5`

10 Problema L: Lugares na Mesa

`Arquivo: lugares.[c|cpp|java]` `Tempo limite: 30 s`

O aniversário de Bilbo está chegando, e Frodo e Sam estão encarregados de todo planejamento da festa! Eles convidaram todos os *hobbits* da Terra-Média para a festa, e todo mundo vai estar sentado em uma única linha em uma mesa de jantar extremamente longa.

No entanto, devido à má comunicação, Frodo e Sam têm cada um independentemente um mapa dos lugares onde todos os *hobbits* vão se sentar na mesa de jantar. Ajude Frodo e Sam descobrir o quão semelhantes são seus mapas dos lugares pela contagem total de pares distintos de *hobbits* que aparecem em ordem diferentes nos dois mapas.

Entrada

O arquivo de entrada contém múltiplos casos de testes. Cada caso de teste começa com uma simples linha contendo um inteiro N ($1 \leq N \leq 100000$) indicando o número de hobbits. As próximas duas linhas representam os mapas de lugares de Frodo e Sam, respectivamente. Cada mapa de lugar é especificado com uma única linha com N palavras alfabéticas únicas; o conjunto de palavras em cada linha é garantido ser idêntico. O final de arquivo é definido por uma linha contendo o número `0`.

Saída

Para cada caso de teste imprima a taxa de diferença entre o mapa de lugares de Frodo e de Sam. A taxa de diferença é definida como o menor número de permutações de lugares entre hobbits vizinhos que faça com que os mapas sejam iguais.

Exemplo de Entrada	**Exemplo de Saída**

`3`
`Frodo Sam Bilbo`
`Sam Frodo Bilbo`
`5`
`A B C D E`
`B A D E C`
`0`
	`1`
	`3`

11 Problema M: Mordor

`Arquivo: mordor.[c|cpp|java]` `Tempo limite: 2 s`

Mordor é uma relíquia da destruição causada por Morgoth, e é formada basicamente por erupções vulcânicas. O seu território é cercado pelas montanas intransponíveis de Ered Lithui (ao norte) e Ephel Dúath (oeste e sul). Existe uma passagem na parte noroeste, entre estas duas montanhas chamada de Cirith Gorgor. Esta passagem é controlada pelo Portão Negro, onde nada entra e nada sai sem a permissão e observação de Sauron. Assim, ninguém consegue entrar ou fugir de Mordor.

Bem, na verdade não é bem assim. Existem passagens secretas, que geralmente são vigiadas por Sauron. Porém, em alguns momentos Sauron precisa desviar sua atenção para tratar de assuntos inesperados, deixando estas passagens sem vigias. Alguns orcs aproveitam estas brechas para tirar umas férias de Mordor, com a permissão e registro dos seus capitães.

Os capitães orcs estão preocupados que alguns destes orcs fujam, ou seja, nunca retornem destas férias. Sendo assim, você foi forçado, digo, contratado por um capitão de Sauron para escrever um programa[1] que calcule quantos orcs ainda não retornaram à Mordor com base em registros de saída e retorno.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é iniciado por um inteiro N ($0 \leq N \leq 1000$) que diz o número de registros, seguido por N números inteiros (entre -100 e 100, inclusive) que informam quantos orcs saíram (número negativo) ou retornaram (positivo) para Mordor. A entrada termina quando $N = 0$, caso que não deve ser tratado.

Saída

Imprima uma linha contendo o número total de orcs que não voltaram para Mordor. Caso o saldo de orcs seja positivo, ou seja, o número de orcs que entraram for maior do que o número de orcs que saíram, algum relatório está errado e você deve imprimir somente a mensagem *erro*. Confira os exemplos abaixo.

Exemplo de Entrada	**Exemplo de Saída**
5 -1 -2 -3 -4 -5 3 1 -3 2 4 -3 4 -2 3 0	15 0 erro

[1] O programa deverá rodar em um Palantír v2.1 turbo, Mordor® edition