1ª Maratona Catarinense de Programação

Prova Principal

Joinville, 15 de junho de 2013

Sevidor BOCA:

`http://10.20.107.205/`
(**Seletiva UDESC**)

`http://10.20.107.207/`
(1ª **MCP**)

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Lucas Hermann Negri (coordenação técnica), Yuri Kaszubowski Lopes, Marlon Fernandes de Alcântara, 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;
- SBC – Sociedade Brasileira da Computação;
- Realização: DCC/UDESC;
- Rutes, pelo empenho neste ano;
- Alguns, muitos outros anônimos.

Conteúdo

1 Problema A: Agora Vai

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

Autor: Yuri Kaszubowski Lopes

Seja bem vindo à primeira Maratona Catarinense de Programação. A organização deseja sucesso ao seu time e esperamos que esta seja uma competição memorável. Todo o *staff* preparou carinhosamente cada detalhe desta prova. Realizamos testes e esperamos que não haja nenhuma inconformidade. Claro que não se pode garantir que não haja falhas na prova, por isso fique atento aos *clarifications* e o utilize em caso de dúvidas. Os organizadores trabalharam duro para proporcionar a vocês uma Maratona realmente ...

Ah! Você quer é saber do problema, o tempo já está correndo, é claro! Bem este primeiro problema esta estritamente relacionado com os preparativos desta primeira Maratona Catarinense. Como alguns de vocês já devem saber, toda a organização foi liderada pelo professor Claudio Cesar de Sá, conhecido e de agora em diante denominado ĈĈ (leia-se "*cêcê*"). Bem, acontece que nosso querido professor ĈĈ é digamos um tanto perfeccionista e com a organização da Catarinense não poderia ser diferente. ĈĈ escreveu alguns dos problemas da prova e acompanhou de perto a elaboração dos demais, cuidou da alocação dos laboratórios, do servidor, de convidar as diversas instituições para este evento e, é claro, teve um total carinho com as camisetas da maratona.

E é justamente sobre as camisetas da maratona que se trata este problema. O professor ĈĈ estava agoniado com o prazo de entrega das camisetas e não era por menos, elas realmente tomaram um certo tempo para ficarem prontas. O motivo foi que ĈĈ contratou o senhor Rod para confeccionar nossas camisetas. O senhor Rod tem uma fabriqueta onde confecciona roupas e ele tem uma característica peculiar, ele gosta de maximizar seus lucros. E por isso as camisetas demoraram um pouco mais, já que o senhor Rod estava realizando os cálculos para cortar os tecidos para as camisetas de maneira ótima, mas sempre ficava na dúvida se seus cálculos estavam corretos. E este fato por sua vez resultou em preocupações para o professor Claudio. Assim, o professor ĈĈ solicitou-me que o primeiro problema da maratona fosse um programa de otimização do corte dos tecidos das camisetas que será doado ao senhor Rod para que não tenhamos mais problemas no próximo ano.

Após uma pesquisa com o senhor Rod constatamos que seus tecidos são comprados em um rolo com a largura exata para a confecção de uma camiseta mas com um enorme comprimento que deve ser cortado. O senhor Rod nos informou que dependendo da posição do corte ele consegue economizar mais ou menos em fio (para juntar os pedaços de tecido), mas que também deve ficar atento ao máximo aproveitamento do rolo de tecido e de que há certas medidas que são inúteis. Dado o tamanho do rolo em metros e uma tabela que relaciona tamanhos de comprimentos válidos com o lucro que este tamanho produz escreva um programa que informe ao senhor Rod qual o lucro máximo que ele pode obter com aquele rolo, ajudando assim o senhor Rod a conferir seus cálculos.

Por exemplo, para um rolo de 200 **metros** e cinco possíveis comprimentos: 90cm, 95cm, 100cm, 105cm e 110cm nos quais os respectivos lucros são R$ 0,91; R$ 0,96; R$ 1,00; R$ 1,05 e R$ 1,10 (veja o primeiro caso do exemplo) o valor máximo que o senhor Rod pode obter é de R$ 202,22. Pois ele produzirá 218 camisas utilizando a medida de 90cm e 4 camisas utilizando a medida de 95cm; assim, $218 \times 0,91 + 4 \times 0,96 = 202,22$ e $218 \times 90cm + 4 \times 95cm \leq 200m$. Observe que o melhor corte pode não lhe obrigar a usar todo o tecido.

Entrada

A entrada é composta de vários casos de testes, cada caso inicia com dois valores l e n indicando o comprimento do rolo que o senhor Rod possui **em metros** e a quantidade de comprimentos válidos respectivamente, com $0 < l, n \leq 200$. Cada uma das próximas n linhas contém dois inteiros c e v que descreve um comprimento válido **em centímetros** e seu respectivo valor respectivamente, com $0 < c \leq 200$ e $0 < c \leq 1000$ **em centavos de Reais**.

Saída

Para cada caso de teste imprima uma linha contendo o valor **em Reais** com duas casas decimais do lucro máximo que o senhor Rod pode obter com este rolo (utilize um ponto e não uma vírgula para separar as casas decimais).

Exemplo de Entrada

```
200 5
90  91
95  96
100 100
105 105
110 110
2 2
100 200
150 375
0 0
```

Exemplo de Saída

```
202.22
4.00
```

Mas afinal qual o motivo do nome deste problema?

Foi exatamente isso que o preocupado professor ĈĈ me perguntou. A fim de finalizar esta prova estava faltando exatamente um problema, já que alguns não passaram pelo crivo do grande ĈĈ . Assim, quando este problema ficou pronto eu falei para mim mesmo: Agora vai! ☺ by Yuri!

2 Problema B: Bytes de Memória

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

Autor: Marlon Fernandes de Alcântara

Lucas trabalha em uma loja de informática e tem um serviço muito importante, verificar a lista de pedidos de memória RAM e analisar quantos pentes serão necessários para suprilas. As memórias RAMs, como todos sabem, possuem uma capacidade que sempre é uma potência de 2 (1, 2, 4, 8, 16, 32...), logo um pedido pode pode requerer um único pente ou uma combinação de pentes distintos.

Todavia, com o aumento da capacidade das memórias RAMs e o crescente números de pedidos, a tarefa de Lucas está levando muito tempo. Por isso, você deve fazer um programa para ajudá-lo.

Entrada

A entrada é composta por vários casos de teste. Cada caso é iniciado com um valor N ($0 < N \leq 1.000.000$, $N = 0$ marca o final da entrada) que indica o número de pedidos do dia, na próxima linha tem-se N números separados por espaço indicando a capacidade (C_i, $0 \leq C \leq 2^{32}$) requerida no i-ésimo pedido.

Saída

A saída deverá ser a quantidade de pentes de memória que serão vendidos no dia de acordo com a lista de pedidos. Quando N for igual a zero é indicado o fim da entrada, que não deve ser processado.

Exemplo de Entrada

```
1
93
2
38 18
3
25 97 2
4
60 44 58 36
0
```

Exemplo de Saída

```
5
5
7
13
```

3 Problema C: Chaveiro Polonês

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

Autor: Yuri Kaszubowski Lopes

Santa Catarina é sem dúvida nenhuma um belíssimo estado, sua colonização feita por diversos povos europeus é uma característica marcante. Aqui se instalaram principalmente imigrantes alemães, italianos e portugueses açorianos; em menor quantidade outros povos europeus como os poloneses (o bom e velho polaco), imigrantes de outros continentes; nativos (indígenas) também se fazem presentes. Todos trouxeram na bagagem sua cultura e costumes, uma vasta culinária e claro cada um possui sua preferência no quesito bebida. Os alemães, por exemplo, não perdem a oportunidade de tomar um *schoppen*[1] de cerveja, os italianos se agarram em uma garrafa de vinho, *Tutto bene*. Já os poloneses gostam de vodca e da cor amarela, mas ao chegarem ao sul do país adotaram a cachaça como bebida preferida, principalmente se for aquela amarelinha, *bardzo dobre*.

Obviamente, como esse povo todo gosta de tomar umas e outras de vez em quando alguns problemas acabam surgindo, principalmente relacionados a perda de objetos. Não há nada mais chato do que perder algo, principalmente se isso for a chave da casa e acordar a *Frau*[2] no meio da noite não é uma tarefa agradável. Pensando nisso Petroski, um polaco de meia idade, aprendeu o ofício de chaveiro. E devido ao grande volume de solicitações ele pretende comprar kits de um novo modelo para destrancar portas que são realmente muito rápidos de serem usados. No entanto, um *kit* não é capaz de destrancar todas as portas. Petroski pediu sua ajuda para que você escreva um programa que determine se um determinado *kit* pode abrir um conjunto de trancas. Isso ajudará Petroski na compra de seus kits e no seu dia a dia, já que ele tem registrado os códigos das trancas de todos os seus principais e rotineiros clientes.

O *kit* é composto de até 5 chaves muito finas, as quais você pode combiná-las para abrir uma porta. Uma tranca necessita de uma combinação exata na altura de cada um de seus pinos ao mesmo tempo, alguns pinos não devem ser movimentados (altura 0 da chave naquela posição) outros tem que ser empurrados totalmente (altura 9 da chave naquela posição). Ao combinar duas chaves, na qual em determinada posição elas tenham altura diferente o resultado é que o pino é empurrado de acordo com o maior valor. Ainda é possível inserir parcialmente a chave, sendo possível, por exemplo, criar diversas combinações com apenas duas chaves. Finalmente, se uma tranca possui 10 posições, uma chave de 8 posições nunca alcançará os últimos dois pinos (posições 1 e 2 da tranca), mas a posição 1 da chave (ponta) pode ser usado neste caso para as posições 3, 4, ...,10 da tranca.

Note que se a posição 1 da chave interfere na posição 1 da tranca, obrigatoriamente a posição 2 da chave vai interferir na posição 2 da tranca e assim por diante. Um mesmo *kit* pode conter chaves de diversos tamanhos.

Entrada

A entrada é composta por vários casos de testes. A primeira linha de cada caso de teste possui dois números k e l, indicando o número de chaves no *kit* e de trancas a serem analisadas respectivamente, para $1 \leq k \leq 5$ e $1 \leq l \leq 30$. As próximas k linhas contêm cada uma a descrição de uma chave do kit, sendo formada por uma cadeia de números de 0 a 9 indicando

[1] Medida alemã equivalente a 300 ml, palavra que com o tempo, no nosso país, acabou passando a ser utilizado com sinônimo de cerveja não-pasteurizada (Chopp)

[2] Mulher em alemão

a altura que a chave empurra o pino, a cadeia inicia na ponta (esquerda) e termina na parte onde se segura a chave (direita). Cada uma das próximas l linhas descrevem uma tranca que deve ser analisada. Cada tranca é descrita por uma cadeia de números de 0 a 9 indicando a altura que o pino naquela posição deve ser empurrado, a cadeia inicia da posição mais interna da tranca (fundo) até a extremidade onde a chave é inserida (frente). Em ambos os casos cada dígito corresponde a uma posição/pino da tranca. Cada cadeia possui de 1 a 20 posições. A entrada termina com $k = l = 0$.

Saída

Para cada caso de teste seu programa deve imprimir uma linha "Caso: #**n**", sem as aspas, onde n é o numero do caso de teste iniciando em 1. Em seguida para cada tranca do respectivo caso de teste seu programa deve imprimir uma linha "**m**: S" caso a m-ésima tranca do caso pode ser aberta pelo *kit* ou "**m**: N" caso contrário, sem as aspas e m é o número que identifica a tranca iniciando em 1.

Exemplo de Entrada

```
2 3
123456789
9876543210
12
1234
123456789
3 2
111111
123456
654321
0011234664
001126555
3 4
111111
123456
654321
0112345
01123456
00112365
00123656
0 0
```

Exemplo de Saída

```
Caso: #1
1: S
2: S
3: S
Caso: #2
1: N
2: N
Caso: #3
1: S
2: N
3: S
4: S
```

4 Problema D: Divisão das Tarefas na Cozinha

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

Fonte original: World Finals 2001, Problem G – Fixed Partition Memory Management
Tradução e adaptação: Claudio Cesar de Sá

Surpresa, a maratona vai continuar após toda esta programação. Sim, um outro desafio em outro lugar. Onde? Na cozinha do Centro de Convivência no prédio ao lado, aquele que foi inaugurado e que a qualquer momento vai funcionar.

Uma cozinha é um lugar repleto de apetrechos, com uma funcionalidade particular, onde há pessoas hábeis que realizam tarefas diversas, desde a compra dos ingredientes, as que descascam batatas e cebolas, as que fazem a limpeza, lavadores de pratos e panelas, as que sabem abrir um traíra, os que sabem escolher uma bebida, provar um vinho, as que sabem cozinhar, as que sabem servir os pratos, as que fazem a decoração, etc. Enfim, a cozinha aceita pessoas com habilidades diversas, pois há muitas tarefas que exigem particularidades. Estas tarefas, demandam habilidades das pessoas e são cumpridas em um determinado tempo. Na tarefa de descascar cebolas em um restaurante húngaro (além de chorar a noite inteira devido a cebola), exige-se pouca habilidade mas com muita demanda de tempo. Contudo, para cozinhar um arenque deve-se ter muita habilidade para não passar do ponto, e isto ocorre em pouco tempo de cozinha.

Imagine que após esta competição, sua equipe de trabalho (até 3 maratonistas) seguirá para a cozinha e, de acordo com as habilidades ou *expertises* individuais, vai realizar uma lista de tarefas. Logo, dada uma lista com diversas tarefas, onde cada uma apresenta um requisito de habilidade e de duração, o seu desafio é distribuir as tarefas para cada um dos membros da sua equipe, a fim de minimizar o tempo para realizar a lista destas tarefas.

As tarefas podem ser distribuídas para os membros da equipe de forma com que elas possam ser realizadas concorrentemente, ganhando-se tempo, pois cada membro só pode trabalhar em uma tarefa em um determinado instante. Porém, este problema é mais difícil do que parece, pois cada tarefa exige uma quantidade mínima de habilidade para ser executada.

Entrada

A entrada é composta por vários casos de testes. Cada caso de teste começa por uma linha contendo um par de inteiros m e n. O número m especifica o número de membros da equipe ($1 \leq m \leq 3$), e n especifica o número de tarefas a serem cumpridas ($1 \leq m \leq 10$). Nota: com até 10 tarefas numa cozinha pode-se preparar um boa feijoada.

A próxima linha contém m inteiros positivos que indicam o valor das habilidades de cada membro do time. As próximas n linhas deste caso de teste, descrevem a habilidade requerida *versus* tempo para realizar cada uma das n tarefas. Cada uma destas n linhas se inicia com um inteiro positivo k ($1 \leq k \leq 10$), o qual descreve uma quantidade de pares de inteiros de: habilidade requerida e o tempo de realização da tarefa (uma tarefa pode ser realizada de formas diferentes, cada forma com seu requisito de habilidade e de tempo). Estes k pares de inteiros seguem o seguinte formato: $s_1\ t_1\ s_2\ t_2\ ...\ s_k\ t_k$ tal que os mesmos satisfazem $s_i < s_{i+1}$ para $1 \leq i < k$. O mínimo de habilidade exigida para a tarefa é s_1. Se o problema é resolvido por um membro do time com habilidade s, onde $s_i \leq s < s_{i+1}$ para algum i, então seu tempo de solução será t_i. Finalmente, se a tarefa é resolvida por algum membro de habilidade igual ou superior a s_k, então seu tempo de execução será t_k. Repare que um membro com menor habilidade pode acabar realizando uma tarefa em um tempo menor do que um membro

com maior habilidade, devido ao uso de diferentes métodos de preparo. Isto significa que: inicialmente procura-se maximizar a habilidade de cada indivíduo (afinal quem sabe cozinhar bem, não deve descascar batatas), contudo, com o critério de minimizar o tempo total da lista de tarefas.

Um par de zeros ao final indica o final dos casos de testes. Todas as tarefas podem ser realizadas por algum membro da equipe.

Saída

Para cada caso de teste, imprima uma linha com o número do caso de teste. Em seguida imprima a média do tempo gasto para a realização das tarefas (média do tempo de término), com dois dígitos à direita do ponto decimal. Na sequência deve-se imprimir o escalonamento das tarefas, uma tarefa por linha, na ordem de apresentação da entrada. Cada tarefa resultará em uma linha com o número da tarefa, o membro da equipe selecionado em resolvê-la (numerado na ordem dada na entrada), o tempo quando o membro começou executá-la, e o tempo de quando ela foi resolvida. Siga os exemplos abaixo para formatação, deixando uma linha em branco para cada caso de teste.

Exemplo de Entrada

```
2 4
40 60
1 35 4
1 20 3
1 40 10
1 60 7
3 5
10 20 30
2 10 50 12 30
2 10 100 20 25
1 25 19
1 19 41
2 10 18 30 42
0 0
```

Exemplo de Saída

```
Caso: 1
Tempo medio sol: 7.75
Tarefa 1 eh realizada pelo membro 2 inicio: 0 fim: 4
Tarefa 2 eh realizada pelo membro 1 inicio: 0 fim: 3
Tarefa 3 eh realizada pelo membro 1 inicio: 3 fim: 13
Tarefa 4 eh realizada pelo membro 2 inicio: 4 fim: 11

Caso: 2
Tempo medio sol: 35.40
Tarefa 1 eh realizada pelo membro 3 inicio: 19 fim: 49
Tarefa 2 eh realizada pelo membro 2 inicio: 0 fim: 25
Tarefa 3 eh realizada pelo membro 3 inicio: 0 fim: 19
```

```
Tarefa 4 eh realizada pelo membro 2 inicio: 25 fim: 66
Tarefa 5 eh realizada pelo membro 1 inicio: 0 fim: 18
```

5 Problema E: Espião Probabilístico

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

Autor: Lucas Hermann Negri

Jaime Bonde, espião brasileiro, tem um trabalho muito perigoso: se infiltrar em quartéis da máfia catarinense em busca de evidências que possam levar à prisão dos chefões do crime organizado. Ao entrar em um quartel, Jaime procura por evidências em todas as salas, movimentando-se pelos corredores menos vigiados para passar despercebido.

Para maximizar sua chance de encontrar evidências sem ser detectado pelos criminosos, Jaime pediu sua colaboração: ele passará para você a estrutura de um quartel do crime e vai perguntar (uma ou mais perguntas por quartel) qual é a chance de ser detectado por um criminoso ao passar de uma determinada sala para outra, assumindo o trajeto ótimo.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste é iniciado por dois inteiros N ($0 < N \leq 200$) e M ($0 < M \leq 10000$), sendo que $N = 0$ e $M = 0$ marcam o final da entrada (N é o número de salas do quartel, e M é o número de corredores). Cada uma das M linhas a seguir descrevem um corredor. A descrição de um corredor é composta por 3 inteiros, `A B C`, onde C ($0 \leq C \leq 100$) é a porcentagem de chance de Jaime ser detectado ao passar pelo corredor que liga as salas A e B (sentido duplo) ($1 \leq A, B \leq N$). Todas as salas podem ser alcançadas por meio dos corredores, e não há mais de um corredor entre um par de salas. Após a descrição dos corredores existe um inteiro Q ($1 < Q \leq N$) que representa o número de perguntas. Cada pergunta é representada em uma linha contendo dois inteiros D P, marcando a sala de origem e a sala de destino ($1 \leq D, P \leq N$).

Saída

Para cada pergunta de cada caso de teste, imprima a porcentagem (arredondada para duas casas decimais), do espião ser detectado no trajeto entre o par de salas em questão (siga a formatação vista no exemplo abaixo). Deixe uma linha em branco após cada caso de teste.

Exemplo de Entrada

```
3 3
1 2 2
2 3 3
1 3 5
1
1 3
5 8
1 2 3
1 3 10
1 4 5
2 3 2
4 3 15
2 5 7
3 5 5
4 5 5
2
1 5
2 4
0 0
```

Exemplo de Saída

```
4.94%

9.69%
7.85%
```

6 Problema F: Falta um Planeta

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

Fonte original: 2004 Stanford Local Programming Contest
Tradução e adaptação: Claudio Cesar de Sá

As coisas *andam* muito estranhas ultimamente. Constatamos distorções da realidade por todo lado (isto é filosófico, deixaremos para o fim da prova). Enfim, até na astronomia tem acontecido coisas estranhas, como planetas que giram de modo circular e não em elipsoidalmente. Sim, aqui tudo é bem comportado, e funcionará em círculos perfeitos.

Seja um sistema solar fictício de uma estrela S, digamos o Sol, um planeta P, e sua lua M. O planeta P orbita em uma trajetória circular perfeita em torno da estrela S com um período de revolução exatamente igual a T dias da terra, e sua lua M também orbita em um círculo perfeito em torno do planeta P com um período de revolução desconhecido. Dada a posição desta lua M em relação a estrela S em três diferentes pontos, o seu objetivo é computar a distância do planeta P até a estrela S.

Para fazer isto, considere um sistema de coordenadas cartesianas com duas dimensões com sua origem centrada na estrela S. Neste cálculo, deve-se assumir que P orbita no sentido anti-horário em torno de S e M também orbita no sentido anti-horário em torno de P, ambos completamente dentro do espaço bi-dimensional $x-y$. Seja (x_1, y_1) defina a posição da lua M num primeiro momento de observação, seja (x_2, y_2) defina a sua posição k_1 dias terra depois, e seja (x_3, y_3) defina a sua posição k_2 dias terra depois da segunda observação.

Entrada

A entrada do arquivo teste contém múltiplos casos. Cada caso de teste consiste de duas linhas. A primeira linha contém 3 inteiros, T, k_1, e k_2, onde $1 \leq T, k_1, k_2 \leq 1000$. A segunda linha contém 6 valores em ponto flutuante, x_1, y_1, x_2, y_2, x_3, y_3. As entradas dos pontos tem sido selecionadas para garantirem uma solução única, tal que a distância final do planeta P até a estrela S sempre estará dentro de um intervalo de `0,1` do inteiro mais próximo. O final do arquivo é definido por uma sequência de três zeros: `0 0 0`

Saída

Para cada caso de teste, o seu programa deve imprimir a distância do planeta P até a estrela S, arredondado para o inteiro mais próximo.

Exemplo de Entrada

```
360 90 90
5.0 1.0 0.0 6.0 -5.0 1.0
307 654 641
441.064 -13.5738 148.546 43.6975 155.747 69.1039
0 0 0
```

Exemplo de Saída

```
5
399
```

7 Problema G: Guarda-*Chuvaville*

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

Autor: Marlon Fernandes de Alcântara

A cidade de Chuvaville é famosa pelo alto índice de chuvas que já vem deixando a população da cidade chateada. A cidade tem um formato quadrado, porém suas dimensões ainda não foram calculadas devido ao seu constante crescimento. O governo da cidade sabe que a população não aguenta mais as constantes chuvas e como as eleições estão próximas eles pensaram em fazer uma cobertura para a cidade.

Para cobrir a cidade está sendo projetada uma estrutura retangular (repare que um quadrado é um retângulo de lados iguais), todavia, nem toda a população gostaria de ter suas residências cobertas, pois algumas pessoas cultivam jardins em suas casas ou pequenas plantações, e de certa forma gostam das chuvas e querem continuar com elas.

Pensando nisto, o governo realizou uma pesquisa com toda a cidade e para cada quadrante descobriram quantas pessoas gostariam da cobertura e quantas pessoas não gostariam, atribuindo números positivos para quem é a favor, e números negativos para quem é contra. Assim, um quadrante com o valor 5 mostra que lá existem 5 pessoas a favor da cobertura a mais do que o número de pessoas que são contra a cobertura.

Escreva um programa para ajudar o governo de Chuvaville a construir a cobertura que agrade o maior número de pessoas. Note que é possível que nenhuma pessoa queira a cobertura, logo esta não será construída (área igual a 0).

Entrada

O arquivo de entrada contém vários casos de testes. Cada caso de teste é iniciado por um inteiro n ($0 \leq n \leq 100$), que indica a dimensão da cidade. Em seguida, tem-se n linhas com n colunas, as quais descrevem o quadrante ou matriz que representa a cidade. Um $n = 0$ indica o término da entrada.

Saída

A saída será um único inteiro por linha. Este valor representa uma soma de valores que estão dentro de uma área da cobertura, que maximize a satisfação da população, dada na matriz original. Quando $n = 0$ não escreva nada.

Exemplo de Entrada

```
2
-12 -50
75 25
4
-68 -32 -25 -80
72   5   33 84
-47 -29 -3 -59
-87 -88 66 -27
0
```

Exemplo de Saída

```
100
194
```

8 Problema H: Hoje é Dia

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

Autor: Yuri Kaszubowski Lopes

Final da tarde de sexta-feira, são 17:42 e você espera ansioso para o término do seu seu horário de estágio, afinal amanhã é o dia da Maratona Catarinense e você, como um bom maratonista, vai cedo para a cama. No entanto seu chefe entra desesperado na sala de desenvolvimento: um cliente ligou dizendo que não consegue atualizar a base de dados com os arquivos de *backup*. Então hoje é dia de ficar até mais tarde pra resolver o problema.

Sua empresa adota uma estranha forma de realizar o *backup* de forma que os seus clientes não tenham acesso a modelagem do sistema. Este *backup* consiste em gerar um arquivo para cada campo do banco de dados onde cada entrada deste campo ocupa uma linha de um arquivo de texto puro. Você foi investigar e notou que o arquivo referente ao campo de telefones era o responsável pelo problema. A função que gerava este arquivo simplesmente não colocava o pulo de linha entre um registro e outro. Após solucionar este problema no código fonte, sua tarefa agora é verificar se é possível reaproveitar o *backup* defeituoso.

Sabemos que os números de telefonem variam a quantidade de dígitos de acordo com o seu prefixo. Desta forma escreva um programa que dada uma descrição dos padrões de números de telefone e o conteúdo do *backup* retorne a quantidade de telefones existentes no arquivo de *backup*. Um padrão é formado por um prefixo numérico seguido de letras x que representam qualquer dígito, por exemplo *12xxx* e *120x*. Você deve assumir que os padrões com prefixo maior são testados primeiramente e que cada prefixo é único (*12xxx* e *12xx* não aparecem em um mesmo caso de teste).

Entrada

A entrada é composta por vários casos de testes. A primeira linha de cada caso de teste possui um número n ($0 < n \leq 8$) indicando o número de padrões. Cada uma das próximas n linhas representa um padrão de telefone com até 16 caracteres. A última linha do caso de teste apresenta o conteúdo do arquivo de *backup* a ser verificado, ele pode ter até 150 caracteres e no mínimo 1 caractere. A entrada termina com $n = 0$;

Saída

Para cada caso de teste imprima em uma linha a quantidade de números de telefones existentes no arquivo de *backup* ou -1 quando o conteúdo do arquivo não for compatível com os padrões.

Exemplo de Entrada

```
3
19x
0xxxxxxxxx
xxxxxxxx
19019119204733701234333401 23
2
19x
xxxx
111119019122221191
1
1xxx
12342345
0
```

Exemplo de Saída

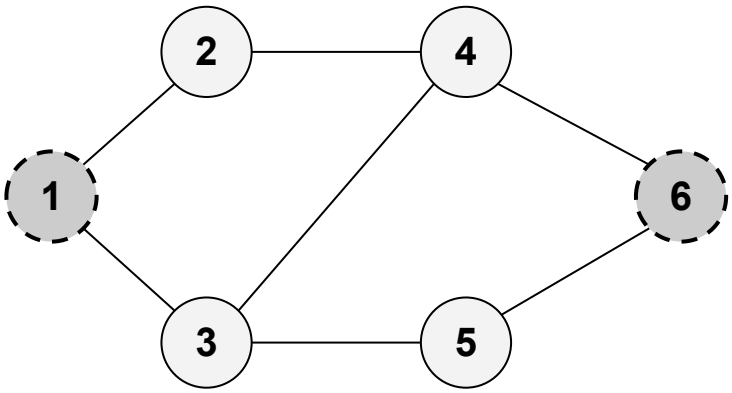
```
5
5
-1
```

9 Problema I: Indiana Jonas

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

Autor: Lucas Hermann Negri

Apesar de ser de conhecimento de poucos, a UDESC Joinville possui um grupo de exploradores que se reúnem de tempos em tempos em busca de relíquias. Indiana Jonas, o líder do grupo, ouviu falar sobre um conjunto de estatuetas localizadas em Corupá, bem pertinho da região das cachoeiras. O grupo foi reunido e partiu para o local. Chegando lá, adentrando na mata, o grupo se deparou com um complexo formado por grandes pilares, com pontes de madeira interligando-os. Foi constatado que os pilares são fortes, mas as pontes, já gastas pelo tempo, não. Quando uma pessoa passa pela ponte, esta se desmancha, ou seja, cada ponte é de uso único, por uma só pessoa. A figura abaixo mostra um exemplo onde no máximo duas pessoas conseguirão chegar no pilar final (6) a partir do pilar inicial (1).



Indiana Jonas e seu grupo se encontram em um pilar inicial, e precisam chegar ao final do complexo de pilares para encontrar as estatuetas. Jonas ligou para você, pedindo sua ajuda. Você foi requisitado para escrever um programa que, dada a descrição do complexo, diga qual é a quantidade máxima de pessoas que conseguirá chegar no pilar final, considerando a fragilidade das pontes.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste é iniciado por dois inteiros, N ($1 < N \leq 300$, 0 somente para o término) e M ($0 < M \leq 5000$), representando o número de pilares e de pontes. Cada uma das M linhas abaixo é composta por dois inteiros A e B ($1 \leq A$, $B \leq 300$), representando a existência de uma ponte ligando os pilares A e B (sentido duplo). O pilar inicial é sempre o 1, e o final é sempre o N. O final da entrada é representado por $N = 0$ e $M = 0$, não devendo ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro representando a quantidade máxima de pessoas que podem chegar no pilar final a partir do pilar inicial.

Exemplo de Entrada

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

Exemplo de Saída

```
1
2
```

10 Problema J: Juz por Letras

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

Fonte original: alguma seletiva da Nova Zelândia
Tradução e adaptação: Claudio Cesar de Sá
Implementação: Lucas Hermann Negri

Um exercício feito na escola primária é o de completar letras que estão faltando numa frase. Com isto o estudante aprende as letras que formam um palavra, bem como leem com mais atenção a frase. Assim, os professores estão solicitando um gerador automático destas frases incompletas, onde sempre faltam duas letras.

Tarefa

Serão fornecidas duas letras que se deseja suprimir numa determinada frase. No lugar destas letras, seu programa deve imprimir o "_" (*underscore*).

Embora o exercício seja trivial, ele tem sua importância pois permite ao professor identificar rapidamente alunos com problemas de dislexia, falta de atenção, etc.

Entrada

A entrada consiste de vários casos de teste, terminando com uma linha contendo apenas # #.

Cada caso de teste se inicia com duas letras minúsculas na mesma linha, separadas por um espaço. Na linha seguinte virá um simples inteiro n ($1 \leq n \leq 100$), o qual indica o número de linhas que serão construídas nas frases que serão substituídas. Nestas n linhas cada uma contém um número máximo de 1 até 255 caracteres.

Saída

A saída de casa caso conterá o número do caso (veja os exemplos abaixo), e as frases com as letras substituídas. Embora as letras fornecidas sejam minúsculas, troque-as também quando elas forem maiúsculas.

Deixe uma linha em branco após cada caso de teste.

Exemplo de Entrada

```
a e
2
Atarefado foi como estive
Esse exemplo eh claro
o h
3
Hoje talvez chova
Que bom que hoje faz frio
Ontem fez calor
# #
```

Exemplo de Saída

```
Caso 1
_t_r_f_do foi como _stiv_
_ss_ _x_mplo _h cl_ro

Caso 2
__je talvez c__va
Que b_m que __je faz fri_
_ntem fez cal_r
```