Passando a Limpo

1ª Seletiva Interna – 2015/1

Servidor BOCA:

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

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

Organização e Realização:

Claudio Cesar de Sá (coordenação geral), Lucas Hermann Negri (coordenação técnica), Ricardo Oliveira (UFPR), Rogério Eduardo da Silva (revisão técnica), Roberto Silvio Ubertino Rosso Jr., Ramon de Oliveira, Diego Buchinger

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**;
- Exemplo de leitura de entradas:

  ```
  Java: (import java.util.Scanner)
  Scanner in = new Scanner(System.in); int integer1 = in.nextInt();

  C: (#include <stdio.h>)
  int integer1; scanf("%d", &integer1);

  C++: (#include <iostream>)
  int integer1; std::cin >> integer1;
  ```

  Exemplo de saída de entradas:

  ```
  Java: System.out.format("%d %d\n", integer1, integer2);
  C: printf("%d %d\n", integer1, integer2);
  C++: std::cout << integer1 << " " << integer2 << std::endl;
  ```

- 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 *Passando a Limpo* é o nosso chamado do que esta nação precisa.

Patrocinador e Agradecimentos

- Conta Azul – Patrocinador oficial do ano de 2015;
- DCC/UDESC;
- Aos bolsistas deste ano pelo empenho;
- Alguns, muitos outros anônimos.

Passando a Limpo

1^a Seletiva Interna da UDESC

11 de abril de 2015

Conteúdo

1 Problema A: Alagados

`Arquivo: alagados.[c|cpp|java]` `Tempo limite: 5 s`

Nas últimas semanas ocorreram fortes chuvas na cidade de Joinville tendo como consequência o alagamento de diversas ruas da cidade, por este motivo os habitantes da cidade não conseguirão percorrer o caminho que normalmente fazem no dia-a-dia (origem–destino). Para contornar este problema, uma empresa de GPS quer oferecer a população um serviço que informe aos usuários a menor distância entre dois pontos quaisquer da cidade, sendo que nenhuma rua do caminho esteja alagada. O primeiro ponto informado pelo usuário será sua localização e o segundo ponto o destino desejado.

Assim, você foi convocado com urgência para desenvolver um algoritmo que será utilizado neste serviço. Sua tarefa é dado o mapa da cidade, as ruas que estão alagadas e uma lista de par de pontos (A_i, B_i) na cidade, informar para cada um dos pontos a menor distância partindo do ponto A_i e chegar no ponto B_i. Logo, será dado um conjunto de pontos (A_i, B_i), representando uma origem e destino respectivamente, e o menor caminho entre estes dois pontos deve ser encontrado, de maneira que respeitem as ruas alagadas no mapa da cidade.

Entrada

A entrada do problema é composta por diversos casos de teste, onde cada caso é composto por: número de pontos na cidade, número de ruas, a distância entre os pontos (distância nas ruas), número de ruas alagadas, os números entre os pontos alagados, quantas origens e destino queremos saber, e quais são estes.

Especificando cada caso de teste inicia com uma linha contendo dois inteiros N e M, sendo $0 < N \leq 100$ o número de pontos na cidade (cada ponto da cidade é identificado de $0...N-1$) e $M \geq 0$ indicando o número de ruas da cidade. Seguem-se M linhas contendo cada uma três inteiros a_i, b_i, c_i representando a existência de um rua entre os pontos a_i e b_i com distância c_i.

A próxima linha contém um inteiro $K \leq \frac{M}{2}$ representando o número de ruas alagadas na cidade. Cada uma das K linhas seguintes contém dois inteiros u_i e v_i indicando que a rua que conecta os pontos u_i e v_i está alagada. A próxima linha contém um inteiro $1 \leq Q \leq \frac{(M-1)*M}{2}$ indicando o número de consultas feitas no serviço, cada uma das Q linhas seguintes contém dois inteiros A_i, B_i sendo A_i a localização do usuário e B_i o destino desejado.

Saída

Para cada consulta realizada você deve imprimir na tela a menor distância entre os pontos A_i e B_i, caso não seja possivel chegar no ponto B_i partindo de A_i imprima -1.

Exemplo de Entrada	**Exemplo de Saída**
2 1 0 1 1 0 1 0 1 6 7 0 2 3 0 3 4 1 3 3 2 3 3 2 5 1 3 4 2 4 5 2 2 3 4 1 3 5 4 5 2 5 0 2 5 1 3 4 0 0	1 2 1 3 -1 6

2 Problema B: *Beleza*, eis a cópia da nossa cerveja!

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

Calcular cadeias ou sequências de DNA está em alta por várias razões práticas e econônicas. Veja, um novo vírus ou bactéria que surge, refletindo numa epidemia, há um interesse em descobrir logo a sequência afim de programar um novo antibiótico que faça frente a esta doença. Calcular cadeias ou sequências de DNA tem finalidades diversas, doenças, agricultura, produção de alimentos, de bebidas, etc.

Até mesmo devido a chopada (considere que tinha vários tipos de cervejas por lá) dos calouros de ontem, após alguns goles, surgiu um dilema: *posso produzir uma cerveja* ***T*** *(target = alvo) a partir de uma cerveja* ***S*** *(source = origem)?* Logo, o nosso problema aqui trata desta produção genética de uma cadeia de DNA em outra. E claro que alguns estudantes queriam conhecer tais regras, afim de produzirem cervejas mais sofisticadas, a partir do mesmo material genético. Será possível?

A evolução genética é semelhante ao desenvolvimento destas cadeias, há um processo aleatório. Este funciona de uma maneira que se assemelha a certas abordagens usadas para obter soluções aproximadas para problemas combinatoriais difíceis. Mas agora, você terá que fazer algo diferente, como dada duas sequências, indicar em quantos passos uma se reproduziu na outra.

Dada uma cadeia ou sequência (*string*) de DNA **S** (*source*) do alfabeto $\{A, C, G, T\}$, encontre o número mínimo de operações de cópias/reproduções necessárias para criar uma outra *string* **T** (*target*). Você pode inverter as cadeias copiadas, e copiar tanto de **S** como de pedaços de sua parcial **T**. Você pode por esses pedaços juntos a qualquer momento. Você pode somente copiar partes contínuas de sua parcial **T**, e todas cadeias copiadas devem ser usadas em sua parte final de **T**.

Veja os exemplos abaixo e detalhadamente o seguinte exemplo:
A partir de **S** = "`ACTG`" crie ou reproduza **T** = "`GTACAATTAAT`"

1. Pegue GT..... pela cópia e inversão de "TG" de **S**.
2. Pegue GTAC..... pela cópia de "AC" de **S**.
3. Pegue GTAC...TA.. pela cópia de "TA" da cadeia parcial **T**.
4. Pegue GTAC...TAAT pela cópia e inversão de "TA" da cadeia parcial **T**.
5. Pegue GTACAATTAAT pela cópia de "AAT" da cadeia parcial **T**.

Observe que com 5 passos a reprodução da cerveja "`ACTG`" gerou a cerveja "`GTACAATTAAT`".

Entrada

A primeira linha da entrada é um inteiro simples, $1 \leq t \leq 100$, o qual é o número total de casos de testes. Em seguida, para cada caso de teste é dado por duas linhas: uma linha com a cadeia **S** de comprimento $1 \leq m \leq 18$, e a linha subsequente com a cadeia **T** de comprimento $1 \leq n \leq 18$.

Saída

Para cada caso de teste a saída é o número de operações necessárias para criar uma cadeia **T** a partir de **S**. Caso não seja possível fazer tal reprodução, imprimir "`IMPOSSIVEL`".

Exemplo de Entrada	**Exemplo de Saída**
5 ACGT GTAC A C ACGT TGCA ACGT TCGATCGA A AAAAAAAAAAAAAAAAA	2 IMPOSSIVEL 1 4 6

Obs: aqui foram 5 casos de testes

3 Problema C: Cartas da Sorte

`Arquivo: cartas.[c|cpp|java]` `Tempo limite: 5 s`

Os cassinos estão para adicionar mais um jogo de cartas nas suas casas de apostas. Este novo carteado tem o nome de *Cartas da Sorte*, e é jogado da maneira descrita abaixo.

O jogo é individual, e o objetivo é fazer o maior número possível de pontos. Três pilhas de cartas (de valor entre 1 e 9) são feitas, cada uma com um mesmo número de cartas que as demais. A cada rodada, o jogador tem a opção de pegar qualquer combinação de cartas que estejam atualmente no topo das pilhas, isto é, pode-se escolher/pegar somente uma carta de uma pilha qualquer, duas cartas ou até três cartas (uma de cada pilha). O jogador ganha, a cada turno, um número de pontos igual a multiplicação dos valores das cartas (se somente uma carta for escolhida, use o valor da carta).

Por exemplo, se ele pegou as cartas 5 e 6, ganhará 30 pontos. Se pegou as cartas 1, 1 e 1, ganhará 1 ponto. Porém, existe uma restrição: o somatório dos valores das cartas deve ser ímpar, e deve ser menor do que 15. O jogo acaba quando nenhuma rodada a mais puder ser realizada.

Sua tarefa é auxiliar os cassinos a verificarem a pontuação máxima que poderia ser obtida a cada jogo, para se ter uma ideia como se poderá lucrar com ele futuramente. Dada a configuração das pilhas de cartas, determine a maior pontuação possível para ela.

Entrada

A entrada é composta por várias instâncias. Cada instância é iniciada por um inteiro N ($0 \leq N \leq 100$), que identifica o número de cartas em cada pilha. A entrada termina quando $N = 0$. Cada uma das N linhas seguintes contém três inteiros A, B e C, que descrevem os valores numéricos das cartas em um nível da pilha ($1 \leq A, B, C \leq 9$). As pilhas são descritas do topo até o fundo.

Saída

Para cada instância, imprima uma linha contendo a maior pontuação possível.

Exemplo de Entrada	**Exemplo de Saída**
2 5 9 6 1 1 1 3 1 2 3 1 2 3 1 2 3 2 8 8 8 9 9 9 0	42 21 0

Exemplo: no primeiro caso, a pontuação de 42 pontos é obtida pegando as cartas nesta sequência: (5, 6), (9), (1), (1) e (1).

4 Problema D: Demonstração Popular

`Arquivo: demonstracao.[c|cpp|java]` `Tempo limite: 5 s`

Há pouco tempo, muitos brasileiros foram as ruas para participar de demonstrações populares. Um grupo demonstrava seu apoio ao governo, enquanto que o outro demonstrava sua insatisfação. O Cerne Interno de Análise (CIA) recebeu a tarefa de analisar os registros fornecidos pela polícia, que contém a quantidade de manifestantes de cada grupo, organizado por local da manifestação.

Porém, o CIA está com sérias dificuldades, pois demitiu o seu único programador como medida para corte de custos. Os registros são muito extensos, e não há ninguém capaz de escrever um programa para agregar os dados (afinal, só sobrou a diretoria).

O CIA teve uma brilhante ideia: ligou para a UDESC e sugeriu a inclusão da tarefa disfarçada como um problema de maratona. Assim, poderá contar com o trabalho gratuito dos competidores!

Sua tarefa é escrever um programa que leia os relatórios policiais e imprima o valor total de manifestantes de cada grupo (grupo contra o governo, grupo a favor do governo).

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste começa com uma linha contendo um inteiro C (0 até 100), que representa o número de cidades onde ocorreram manifestações. Na sequência encontram-se mais C linhas, cada uma contendo três campos separados por espaços simples: o nome da cidade (até 50 caracteres), o número de manifestantes que apoia o governo e o número de manifestantes contra o governo (valores entre 0 e 1.000.000 cada). Porém, os valores numéricos estão representados com um ponto de agrupamento de milhares (vide exemplo de entrada).

A entrada termina com $C = 0$, caso que não deve ser processado. Nota: o inteiro C não é representado com ponto e agrupamento de milhares.

Saída

Para cada caso de teste, imprima uma linha contendo contendo o número total de manifestantes, o número de manifestantes a favor do governo e o número de manifestantes contra o governo. Os valores devem ser impressos separados por espaço, e os números devem utilizar o ponto de agrupamento de milhares.

Exemplo de Entrada	**Exemplo de Saída**
3 curitiba 100.000 1.000.000 joinville 20.000 5.000 aracaju 500 500 2 alabama 2 1 taio 0 0 0	1.126.000 120.500 1.005.500 3 2 1

5 Problema E: Estressantes Mosquitos

`Arquivo: estressantes.[c|cpp|java]` `Tempo limite: 5 s`

Não são os elefantes, mas sim os mosquitos que encomodam muita gente. Recorda daquele *picnic* que foi estragado por causa dos mosquitos? Então, agora é a sua hora para se vingar. Em um novo *picnic*, você decidiu se precaver contra os mosquitos, mas acabou esquecendo dos equipamentos de extermínio de mosquitos. O único equipamento disponível é a tigela que você havia guardado o purê de batata.

Olhando para a toalha de *picnic*, você percebeu uma grande quantidade de mosquitos, só esperando para atacar você... mas não será desta vez! Equipado com a tigela, você irá se defender!

A sua tarefa é determinar a maior quantidade de mosquitos que podem ser pegos por um único golpe de tigela (colocando-a rapidamente na mesa acima dos mosquitos, com a boca para baixo). Você tem a disposição o diâmetro da tigela e as posições dos mosquitos; aqui *posições*, leia-se: coordenadas. Um mosquito que fique exatamente na borda da tigela também é considerado pego.

Entrada

A entrada é iniciada por um inteiro N ($1 \leq N \leq 1.000$) que marca o número total de casos de teste. Cada caso de teste é iniciado por uma linha em branco seguida por uma linha com um inteiro M ($1 \leq M \leq 32$) correspondente ao número de mosquitos e um valor real D ($0 \leq D \leq 200$) que corresponde ao diâmetro da tigela. As próximas M linhas contém a posição de cada mosquito, ma forma de coordenada (números reais) x y, onde $-100 \leq x, y \leq 100$. Entre um caso e outro, há uma linha em branco.

Saída

Para cada caso de teste, imprima o maior número de mosquitos que podem ser pegos com um único golpe de tigela. Assuma que a resposta não vai mudar se o diâmetro da tigela aumentar em até 10^{-5}.

Exemplo de Entrada	**Exemplo de Saída**
2 4 1.5 1.0 3.75 3.0 1.0 1.0 2.25 1.5 3.0 8 3.0 -1.0 3.0 -1.0 2.0 -2.0 1.0 0.0 1.0 1.0 0.0 1.0 -1.0 2.0 -2.0 3.0 -1.0	3 4

6 Problema F: Fila do RU

`Arquivo: fila.[c|cpp|java]` `Tempo limite: 5 s`

A fila no RU (restaurante universitário) é um problema recorrente em muitas universidades públicas. Por algum motivo, a demanda sempre aumenta mais do que a oferta: se o RU duplica de tamanho, a fila triplica!

Para piorar a situação, alguns estudantes gostam de *furar* a fila. Ao encontrar um amigo na fila, estes furões simplesmente entram na fila, ao lado do amigo, desrespeitando todos que estão esperando atrás dele na fila[1].

Inicialmente, todos os estudantes na fila tem um nível de impaciência igual a 0. Quando alguém *fura* a fila, todos que estão atrás dele tem seu nível de impaciência incrementado em 1. Sua tarefa é, dado o número inicial de estudantes na fila, o número de furões e uma lista de posições onde ocorreram os furos, determinar o nível final de impaciência de cada estudante. Assuma que a fila está parada (o RU ainda não abriu) e que um furão não entra de fato na fila, mas sim fica ao lado do amigo (compartilha a mesma posição).

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste inicia com um inteiro E, ($1 \leq E \leq 1.000.000$) que é o número de estudantes na fila ($E = 0$ indica o término da entrada). A próxima linha conterá um inteiro contendo o número de furões F ($1 \leq F \leq 1.000.000$), seguido por F inteiros (separados por espaço), onde cada inteiro corresponde a uma posição válida na fila (índices de 1 até E).

Saída

Para cada caso de teste, imprima uma linha contendo o valor de impaciência de cada estudante, separando cada valor por espaços (sem espaço extra no final da linha).

Exemplo de Entrada	**Exemplo de Saída**
5 2 1 2 3 5 1 1 2 3 1 0	0 1 2 2 2 0 3 4

[1] Já escutei um furão reclamar da corrupção no governo, acredita? Claro, uma estratégia de enganar duplamente os demais na fila.

7 Problema G: Gabarito

`Arquivo: gabarito.[c|cpp|java]` `Tempo limite: 5 s`

O jovem empreendedor CC está esperançoso neste ano de 2015, pois ele acaba de inaugurar sua nova empresa: ACM – Associação dos esCaneadores de Materiais. As expectativas são grandes porque durante este ano serão realizados diversos concursos públicos, vestibulares e diversos outros tipos de provas que precisarão de correção automatizada.

O primeiro contrato da ACM, a empresa de CC, é com a UDESC – União Didática Esportiva de Santa Catarina, instituição que promoveu diversos concursos públicos e precisa de resultados urgentemente. O método avaliativo desta última instituição, entretanto, é um pouco diferente: para cada resposta correta, o candidato recebe 1 (um) ponto; para cada resposta errada ou com múltiplas marcações de alternativas (duas ou mais respostas), o candidato perde 0,5 (meio) ponto; e caso uma pergunta não seja respondida, ou seja, nenhuma alternativa assinalada, o candidato não ganha e nem perde ponto. Para a UDESC, os candidatos que conseguirem uma pontuação igual ou superior à metade do número de questões da prova são considerados aprovados (aptos a prosseguir para a segunda fase). Já os candidatos que tiverem pontuação menor do que a metade do número de questões da prova são considerados reprovados.

Para corrigir estas provas objetivas, o jovem CC configurou as suas máquinas escaneadoras de modo que elas façam a leitura de todos os campos das alternativas de todas as perguntas. Quando um espaço está preenchido, a escaneadora lê um valor inteiro entre 0 (preto) e 128 (cinza). Quando o espaço não está preenchido (ou foi mal preenchido), a escaneadora lê um valor entre 129 e 255 (preto). Até aí tudo bem. Mas CC tem um problema: ele não sabe programar! Ajude este jovem empreendedor a se tornar milionário no ramo das máquinas escaneadoras e quem sabe ele não se lembre do seu primeiro programador quando ele estiver em Las Vegas.

Entrada

A entrada é composta por um valor inteiro inicial P ($1 \leq P \leq 50$) que indica o número de provas a serem corrigidas. Depois, para cada caso de teste/prova (P_i) são informados outros três números inteiros: Q ($1 \leq Q \leq 100$), A ($2 \leq A \leq 10$) e N ($2 \leq N \leq 250$), representando o número de questões da prova P_i, o número de alternativas de cada questão da prova P_i e o número de pessoas que realizaram a prova P_i (candidatos), respectivamente. Na próxima linha são informados Q caracteres separados por um único espaço, representando o gabarito da prova P_i. As alternativas são nomeadas por letras do alfabeto, iniciando pela letra 'a' (ou seja, a primeira alternativa é a letra 'a', a segunda alternativa é a letra 'b', e assim sucessivamente).

As linhas subsequentes são formadas pelos dados de leitura das máquinas escaneadoras para cada alternativa de cada pergunta da prova P_i. Para cada grupo de Q linhas são informados A valores inteiros (entre 0 e 255) representando a leitura da folha de resposta do candidato N_i. Logo, o primeiro grupo de Q linhas se refere aos dados do primeiro candidato; o segundo grupo de Q linhas se refere aos dados do segundo candidato; e o e-nésimo grupo de linhas se refere aos dados do e-nésimo candidato.

Saída

Para cada entrada de prova, deve ser impressa uma linha informativa: "`Prova #i`", onde i identifica o número da prova. Nas linhas a seguir devem ser mostrados uma linha informativa

"`Candidato #j: `", onde j identifica o número do candidato, em sequência (na mesma linha) deve ser mostrada a nota final do candidato, sempre com uma casa decimal após a vírgula e a sua situação "`Aprovado`" ou "`Reprovado`". Os resultados de duas provas distintas devem ser separados por uma linha em branco no final. Ao final, deixe uma linha em branco também.

Exemplo de Entrada	**Exemplo de Saída**

`2`
`5 4 3`
`a b c d d`
`0 255 255 255`
`240 0 255 200`
`201 200 0 211`
`213 255 255 0`
`255 255 255 0`
 	
`0 150 255 255`
`255 255 255 255`
`255 255 0 255`
`0 200 200 200`
`200 200 200 0`
 	
`0 25 200 200`
`255 129 255 255`
`255 200 200 200`
`200 200 200 200`
`0 255 255 0`
 	
`7 2 1`
`a a b a b b b`
`0 255`
`0 255`
`0 255`
`0 255`
`200 0`
`200 0`
`255 0`
	`Prova #1`
	`Candidato #1: 5,0 Aprovado`
	`Candidato #2: 2,5 Aprovado`
	`Candidato #3: -1,0 Reprovado`
	
	`Prova #2`
	`Candidato #1: 5,5 Aprovado`

8 Problema H: Hambúrguer

`Arquivo: hamburguer.[c|cpp|java]` `Tempo limite: 5 s`

Humberto e seu irmão Hugo estão realizados dietas opostas: Humberto está acima do peso, enquanto que Hugo quer engordar. Os dois irmãos gostam muito de comer Hambúrgueres, e terão que adaptar as respectivas dietas para ajustar o consumo diário de calorias. Ajude os irmãos a calcular as calorias consumidas por meio de hambúrgueres.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste contém um inteiro C ($100 \leq C \leq 2000$), que marca a quantidade de calorias por hamburguer, e os inteiros A e B ($0 \leq A, B \leq 10$) que marcam a quantidade de hambúrgueres consumidos por Humberto e Hugo. A entrada termina com $C = A = B = 0$.

Saída

Para cada caso de teste, imprima uma linha com a quantidade de calorias consumidas por Humberto e Hugo, separadas por espaço conforme o exemplo abaixo.

Exemplo de Entrada	**Exemplo de Saída**

`200 3 1`
`500 2 3`
`0 0 0`
	`600 200`
	`1000 1500`

9 Problema I: Ilhados

`Arquivo: ilhados.[c|cpp|java]` `Tempo limite: 5 s`

As últimas chuvas em Joinville tem alagado certas regiões da cidade, tornando ilhadas algumas destas regiões. Logo, se a defesa civil da cidade detectar que alguma região da cidade se tornou inacessível, um alerta vermelho terá que ser disparado.

A cidade é representada por uma matriz $N \times M$ onde cada célula da matriz é uma região da cidade, se a célula conter o valor 1 significa que esta região não está alagada e 0 significa que a região está alagada. Cada região está diretamente conectada as regiões adjacentes a ela (acima, abaixo, esquerda e direita). Logo, pela diagonal não há conexão. Claro, que se no Brasil as rotatórias fossem comum, a exemplo da França, teríamos que rever o problema.

Sua tarefa é ajudar a defesa civil a encontrar regiões não-alagadas que estão isoladas no mapa da cidade. Uma região é considerada isolada quanto a partir desta não for possível chegar em alguma outra região (também não alagada). Estes isolamento é conhecido como *ilhados*, e esta é uma situação de emergência. Lembrando que para ir de uma região a outra, somente é possivel percorrer por regiões não-alagadas.

Entrada

A entrada consiste de diversos casos de teste, cada caso de teste inicia com uma linha contendo dois inteiros $1 \leq N \leq 1000$ e $1 \leq M \leq 1000$. Esses inteiros representam a quantidade de linhas e colunas da matriz, repectivamente. Seguem-se então N linhas, cada uma contendo M inteiros a_{ij} representando o estado da região i j. Cuidado que os M inteiros presentes na linha estão sem espaço. A entrada termina quando `N = M = 0`.

Saída

Para cada caso de teste, caso o mapa não apresente situação de perigo (não existem regiões ilhadas), imprima em uma linha a situação para população civil: `normal`, e caso tenha uma ou mais regiões isoladas, imprima: `isolados`. No caso de `isolados` será disparado um alarme a população, mas, isto não faz parte do problema.

Exemplo de Entrada	**Exemplo de Saída**
2 7 1110111 1011100 4 6 110111 001111 111111 110111 2 4 1100 0011 0 0	normal isolados isolados

10 Problema M: Multiplicação Legal

`Arquivo: multiplica.[c|cpp|java]` `Tempo limite: 5 s`

Com a chopada dos calouros de ontem, alguns tomaram além da conta. Provocando assim uma *amnésia temporária*, em que até a multiplicação ficou em segundo plano. Mas, professores e maratona, tornam esta mistura bem temperada. Assim, uma releitura muita interessante de como ensinar a multiplicação para este pessoal é dada pelo processo explicado abaixo.

Um exemplo seria quando multiplicarmos $345 \times 56 = 19320$ como dado abaixo, usando um reticulado de 2 linhas e 3 colunas, onde os operandos aparecem contornando multiplicações parciais, veja o quadro formado:

```
+---------------+
|   3   4   5   |
| +---+---+---+ |
| |1 /|2 /|2 /| |
| | / | / | / |5|
|1|/ 5|/ 0|/ 5| |
| +---+---+---+ |
|/|1 /|2 /|3 /| |
| | / | / | / |6|
|9|/ 8|/ 4|/ 0| |
| +---+---+---+ |
|/ 3 / 2 / 0    |
+---------------+
```

O primeiro operando, 345, é exibido na parte superior do reticulado (um tipo de *grid*), onde cada dígito é centralizado horizontalmente acima da sua coluna neste reticulado. O segundo operando, 56, é exibida junto lado direito com cada dígito centralizado verticalmente no centro da sua linha neste reticulado. Por exemplo, o número 30 é representado numa única célula deste reticulado, dado por:

```
+---+
|3 /|
| / |
|/ 0|
+---+
```

Este representa o produto do dígito do primeiro operando que está acima da linha de coluna e o dígito do segundo operando que está à direita da sua linha. No nosso exemplo, esta célula representa o produto $5 \times 6 = 30$ que resulta quando multiplicando o 5, do 345, e o 6 do 56. Observe que o dígito da dezena é colocado na parte superior esquerda da célula, e a parte da unidade, o dígito é escrito na parte inferior à direita.

O produto total é então calculado pela soma ao longo das diagonais do reticulado que representam os mesmos valores de lugar do resultado. Por exemplo, no nosso primeiro problema a 19320 produto foi calculada como:

1's dígito da unidade = 0
10's dígito da dezena $= 5 + 3 + 4 = 12$, então resulta em 2 dezenas com o 1 como transporte para centenas
100's dígito da centena $= (1\mathit{carry}) + 2 + 0 + 2 + 8 = 13$, então resulta em 3 centas com o 1 como transporte para milhar
1000's dígito do milhar $= (1\mathit{carry}) + 2 + 5 + 1 = 9$

10000's dígito da dezena de milhar = 1

O produto resultante é colocado com o dígito das unidades na célula à direita do reticulado e, dependendo do seu comprimento, com os dígitos mais significativos seguindo para o lado esquerdo do reticulado. Cada dígito do produto final aparece perfeitamente alinhado com o somatório das diagonais correspondentes.

Para fornecer uma visão estética, usamos uma série de caracteres menos (`-`) para linhas horizontais, e o caracter *pipe* (`|`) para linhas verticais e o caracter barra (`/`) para linhas diagonais. Além disso, usamos um caracter mais (`+`), onde quer que se encontre uma linha horizontal e vertical. Cada reticulado de multiplicação é posteriormente "*emoldurado*"por uma fronteira externa. Há uma linha que contém o primeiro operando o qual é o limite entre a extremidade superior e a linha superior da reticulado, e uma linha entre a parte inferior do reticulado e a borda inferior, que contém uma parte do produto resultante. Existe uma coluna de entre o primeiro `|` e a borda esquerda do reticulado interior, o qual pode conter uma parte do produto resultante, e depois segue uma coluna de borda à direita do reticulado interno, mas antes do caracter mais à direita `|`, este contém o segundo operando. Se o produto não for longo o suficiente para envolver o canto inferior esquerdo, a coluna entre a borda esquerda e à margem esquerda do reticulado, ficam contendo apenas espaços. (Mais adiante, veja o exemplo 3×3.)

Os zeros à esquerda devem ser exibidos dentro de células do reticulado na estrutura da célula. Mas zeros à esquerda nunca devem ser exibidos no produto final, nem deve haver um caracter de barra (`/`) antes do primeiro dígito do produto. Por exemplo, considere o produto de $12 \times 27 = 324$ abaixo:

```
+-----------+
|   1   2   |
| +---+---+ |
| |0 /|0 /| |
| | / | / |2|
| |/ 2|/ 4| |
| +---+---+ |
| |0 /|1 /| |
| | / | / |7|
|3|/ 7|/ 4| |
| +---+---+ |
|/ 2 / 4    |
+-----------+
```

Observe que no reticulado superior à direita, o produto $2 \times 2 = 04$ é exibida com o zero para o dígito das dezenas. No entanto, não há zeros no dígito do milhar no produto resultante 324; nem existe qualquer barra apresentada acima o dígito 3 deste produto.

Entrada

A entrada contém um ou mais casos de testes. Cada caso de teste contém dois inteiros positivos, A e B, tal que $1 \leq A \leq 9999$ e $1 \leq B \leq 9999$. O último conjunto de dados será seguido por uma linha contendo `0 0`.

Saída

Para cada conjunto de dados, produza o *grid* que ilustra como a multiplicação dos dois números funciona, usando esta técnica de reticulado. Veja outros exemplos abaixo:

Exemplo de Entrada

```
345 56
12 27
1 68
9999 7
3 3
0 0
```

Exemplo de Saída

```
+---------------+
|   3   4   5   |
| +---+---+---+ |
| |1 /|2 /|2 /| |
| | / | / | / |5|
|1|/ 5|/ 0|/ 5| |
| +---+---+---+ |
|/|1 /|2 /|3 /| |
| | / | / | / |6|
|9|/ 8|/ 4|/ 0| |
| +---+---+---+ |
|/ 3 / 2 / 0    |
+---------------+
+-----------+
|   1   2   |
| +---+---+ |
| |0 /|0 /| |
| | / | / |2|
| |/ 2|/ 4| |
| +---+---+ |
| |0 /|1 /| |
| | / | / |7|
|3|/ 7|/ 4| |
| +---+---+ |
|/ 2 / 4    |
+-----------+
+-------+
|   1   |
| +---+ |
| |0 /| |
| | / |6|
| |/ 6| |
| +---+ |
| |0 /| |
| | / |8|
|6|/ 8| |
| +---+ |
|/ 8    |
+-------+
+-------------------+
|   9   9   9   9   |
| +---+---+---+---+ |
| |6 /|6 /|6 /|6 /| |
| | / | / | / | / |7|
|6|/ 3|/ 3|/ 3|/ 3| |
| +---+---+---+---+ |
|/ 9 / 9 / 9 / 3    |
+-------------------+
+-------+
|   3   |
| +---+ |
| |0 /| |
| | / |3|
| |/ 9| |
| +---+ |
|  9    |
+-------+
```

Observe: Estas tabelas devem ser precisamente formatadas dentro destas áreas delimitadas, fornecidas nestes exemplos. Os erros em que o resultado pode estar certo, mas havendo espaços em branco, a mais ou a menos, será considerado como *Presentation Error*. Os demais erros serão reportados como *Wrong Answer*.