acm International Collegiate Programming Contest 2006 IBM event sponsor

Maratona de Programação da SBC 2006

Sub-Regional Brasil do ACM ICPC

9 de Setembro de 2006

(Este caderno contém 8 problemas; as páginas estão numeradas de 1 a 16, não contando esta página de rosto)

Sedes Regionais

Região Centro-Oeste

- Universidade de Brasília, Brasília, DF
- UNAES, Campo Grande, MS
- Universidade Estadual do Mato Grosso do Sul, Dourados, MS

Região Nordeste

- Unifor, Fortaleza, CE
- Universidade Federal do Rio Grande do Norte, Natal, RN
- AESO, Olinda, PE
- Unime, Salvador, BA
- Universidade Federal do Maranhão, São Luís, MA

Região Sul

- Unioeste, Cascavel, PR
- URI, Erechim, RS
- Unisul, Florianópolis, SC
- Universidade Federal do Rio Grande do Sul, Porto Alegre, RG
- FURG, Rio Grande, RS

Região Sudeste

- Universidade Federal de Minas Gerais, Belo Horizonte, MG
- Metrocamp, Campinas, SP
- Faculdades Módulo, Caraguatatuba, SP
- Universo, Juiz de Fora, MG
- PUC Poços de Caldas, Poços de Caldas, MG
- Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ
- Uninove, São Paulo, SP
- Unitau, Taubaté, SP
- Unitri, Ubertlândia, MG
- Centro Universitário de Vila Velha, Vila Velha, ES

Região Norte

- Centro Universitário do Pará, Belém, PA
- Faculdade SEAMA, Macapá, AP
- Universidade Federal de Tocantins, Palmas, TO

Promoção: Sociedade Brasileira de Computação
Patrocínio: Fundação Carlos Chagas – IBM – Ci&T – Microsoft – UOL

Problema A
Circuito Bioquímico Digital

Nome do arquivo fonte: `circuito.c`, `circuito.cpp`, `circuito.java` *ou* `circuito.pas`

Um circuito bioquímico digital (CBD) é um artefato composto de um conjunto de *pontos de processamento*. Cada ponto de processamento é constituído por um minúsculo receptáculo para reagentes bioquímicos, feito de um substrato biológico que se comporta como um micro-circuito eletrônico digital. Dependendo do estado da reação no receptáculo, o substrato gera dois níveis de voltagem. Um leitor acoplado ao CBD é capaz de realizar a leitura de todos os pontos de processamento de um CDB num dado instante, interpretando os dois níveis de voltagem como 0 ou 1.

Um experimento com o CBD é realizado da seguinte maneira. Os pontos de processamento são carregados com as substâncias de interesse e reagentes apropriados e, a cada intervalo fixo de tempo (tipicamente alguns milisegundos), os pontos de processamento são lidos. Assim, o experimento resulta em uma seqüência de conjuntos (vetores) de bits, cada vetor correspondendo a uma medição do CBD.

Uma seqüência ininterrupta de bits 1 de um mesmo ponto de processamento ao longo do tempo é denominada de *palito*. O *comprimento* de um palito é o número de bits 1 que o compõe (note que o comprimento dos palitos de um experimento pode variar entre um e o número de medições efetuadas). Uma característica importante de um experimento com o CBD é a quantidade e o comprimento dos palitos gerados. A figura abaixo mostra o resultado de um experimento realizado com um CBD de seis pontos de processamento, em que foram efetuadas quatro medições, contendo três palitos de comprimento um, um palito de comprimento dois e um palito de comprimento quatro.

```
0 1 0 1 1 0
0 0 0 1 0 0
0 1 0 1 0 1
0 1 0 1 0 0
```

Você foi contratado para escrever um programa que determine, dado o resultado de um experimento, quantos palitos de comprimento igual ou maior do que um certo valor foram gerados.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém três inteiros P, N e C que indicam respectivamente o número de pontos de processamento ($1 \leq P \leq 1000$), o número de medições efetuadas ($1 \leq N \leq 1000$) e o comprimento mínimo de palitos de interesse ($1 \leq C \leq N$). Cada uma das próximas N linhas contém seqüências de P dígitos $\{0, 1\}$, separados por um espaço em branco. O final da entrada é indicado por $P = N = C = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma única linha da saída, contendo o número de palitos de comprimento maior ou igual a C produzidos pelo experimento.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
2 2 2 1 1 1 1 4 5 3 0 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0	2 2

Problema B
O Problema da Parada

Nome do arquivo fonte: `parada.c`, `parada.cpp`, `parada.java` *ou* `parada.pas`

O Problema da Parada (*The Halting Problem*) é um problema de decisão clássico da Ciência da Computação que consiste, basicamente, em determinar se um dado programa sempre vai parar (ou seja, terminar sua execução) para uma dada entrada arbitrária ou se vai executar infinitivamente. Alan Turing provou, em 1936, que é impossível resolver o problema da parada generalizando para qualquer par programa-entrada. Neste problema, porém, dada a descrição de uma linguagem simples, um programa escrito nessa linguagem e uma entrada para esse programa, você deve determinar se o programa dado pára com a entrada dada e, em caso positivo, qual a saída produzida.

Esta linguagem só trabalha com números inteiros de `0` a `999` (inclusive). Sendo assim, o sucessor de `999` é `0`, e o antecessor de `0` é `999`. Além disso, ela possui dez variáveis (`R0` a `R9`), sendo que a `R0` sempre é atribuído o valor de chamada do programa (ou seja, o parâmetro de entrada) e a `R9` é sempre atribuído o valor de saída (o retorno). No início da execução do programa, é atribuído o valor `0` a todas as variáveis, com exceção de `R0` que recebe o parâmetro de entrada.

As operações básicas são atribuição (`MOV`), soma (`ADD`), subtração (`SUB`), multiplicação (`MUL`), divisão inteira (`DIV`) e resto da divisão inteira (`MOD`). Todas essas operações têm a sintaxe `COMANDO OPERANDO1,OPERANDO2` (sem espaços entre a vírgula e os operandos), onde `COMANDO` é uma dessas operações, `OPERANDO1` é uma das 10 variáveis (`R0` a `R9`) e `OPERANDO2` pode ser uma das 10 variáveis ou um valor inteiro (entre `0` e `999`). Todas as operações modificam o valor de `OPERANDO1`, sendo assim `MOV R4,100` é o equivalente a atribuir o valor `100` a `R4`, enquanto que `MUL R3,R8` é o equivalente a multiplicar `R3` por `R8` e atribuir o resultado a `R3`. A operação `DIV`, assim como a `MOD`, retornam `0` (zero) se `OPERANDO2` for `0` ou se a variável equivalente tiver valor `0`. Ou seja, `DIV R4,0` é o equivalente a `MOV R4,0`. Por divisão inteira, entendemos a parte inteira do quociente da divisão (sem a parte fracionária). Por exemplo, a divisão inteira de `7` por `2` é `3` (sendo o resto `1`).

Existem seis comandos de fluxo de decisão: `IFEQ` (se igual), `IFNEQ` (se diferente), `IFG` (se maior), `IFL` (se menor), `IFGE` (se maior ou igual) e `IFLE` (se menor ou igual). A sintaxe de todos eles é `COMANDO OPERANDO1,OPERANDO2` (sem espaços entre a vírgula e os operandos), onde `OPERANDO1` e `OPERANDO2` podem ser variáveis (`R0` a `R9`) ou valores inteiros (entre `0` e `999`). Assim, o comando `IFEQ R4,123` é o equivalente a testar se `R4` é igual a `123`. Caso a condição testada seja verdadeira, o programa continua a executar normalmente a linha subseqüente ao comando de decisão. Caso a condição seja falsa, o programa passa a executar a linha subseqüente ao `ENDIF` mais próximo. **Todos** os comandos de decisão devem ter um comando `ENDIF` correspondente.

Finalmente, existem os comandos `CALL` e `RET`, ambos com a sintaxe `COMANDO OPERANDO`, onde `OPERANDO` é uma variável (`R0`..`R9`) ou valor direto (entre `0` e `999`). O comando `CALL` chama o próprio programa novamente, passando `OPERANDO` como parâmetro de entrada, ou seja, atribuindo o valor de OPERANDO à variavel R0. Já `RET` termina a execução do programa, retornando o valor de `OPERANDO` como o resultado de saída. **A última linha do programa sempre será um comando** `RET`. Observe que, caso o programa chame a si mesmo através do comando `CALL`, quando a execução voltar, o valor de `R9` vai estar alterado com o valor retornado pelo programa. Note também que **todas** as variáveis (`R0`..`R9`) são *locais*, ou seja, uma chamada subseqüente ao programa não pode alterar os valores guardados nas variáveis da

instância anterior, com exceção, naturalmente, do valor de `R9` que recebe o retorno da instância chamada.

O exemplo a seguir ilustra um programa que calcula o fatorial de um número.

linha	comando
1	`IFEQ R0,0`
2	`RET 1`
3	`ENDIF`
4	`MOV R1,R0`
5	`SUB R1,1`
6	`CALL R1`
7	`MOV R2,R9`
8	`MUL R2,R0`
9	`RET R2`

1a linha: Verifica se o valor de `R0` vale `0`, caso positivo, executa a próxima linha, caso contrário, pula para a 4a linha (`ENDIF` mais próximo).
2a linha: Retorna `1` como saída do programa.
3a linha: Marca o fim do bloco de decisão iniciado na primeira linha.
4a linha: Atribui o valor de `R0` a `R1` (`R1` ← `R0`).
5a linha: Diminui `1` de `R1` (`R1` ← `R1 - 1`).
6a linha: Chama o programa passando `R1` como parâmetro de entrada.
7a linha: Guarda o valor de `R9` (retornado pela chamada anterior) em `R2` (`R2` ← `R9`)
8a linha: Multiplica o valor de `R2` por `R0` (`R2` ← `R2 * R0`)
9a linha: Retorna o valor de `R2` como saída do programa.

A tabela seguir traz um resumo dos comandos para referência:

comando	sintaxe	significado
`MOV`	`MOV OP1,OP2`	`OP1 ← OP2`
`ADD`	`ADD OP1,OP2`	`OP1 ← OP1 + OP2`
`SUB`	`SUB OP1,OP2`	`OP1 ← OP1 - OP2`
`MUL`	`MUL OP1,OP2`	`OP1 ← OP1 * OP2`
`DIV`	`DIV OP1,OP2`	`OP1 ← OP1 / OP2`
`MOD`	`MOD OP1,OP2`	`OP1 ← OP1 % OP2`
`IFEQ`	`IFEQ OP1,OP2`	`if OP1 == OP2`
`IFNEQ`	`IFNEQ OP1,OP2`	`if OP1 != OP2`
`IFG`	`IFG OP1,OP2`	`if OP1 > OP2`
`IFL`	`IFL OP1,OP2`	`if OP1 < OP2`
`IFGE`	`IFGE OP1,OP2`	`if OP1 >= OP2`
`IFLE`	`IFLE OP1,OP2`	`if OP1 <= OP2`
`ENDIF`	`ENDIF`	Marca fim do bloco de execução condicional
`CALL`	`CALL OP`	Chama o programa com `OP` como entrada
`RET`	`RET OP`	`return OP`

Entrada

A entrada contém vários casos de teste. Cada caso de teste se inicia com dois inteiros, L e N, representando respectivamente o número de linhas do programa ($1 \leq L < 100$) e o valor do parâmetro de entrada do programa ($0 \leq N < 1000$). As L linhas seguintes contêm o programa. Pode-se assumir que ele está sempre sintaticamente correto de acordo com as regras definidas acima. Todos os comandos (bem como o nome das variáveis) só conterão letras maiúsculas. O final da entrada é marcado pelo caso em que $L = N = 0$ e não deve ser processado.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, seu programa deve produzir uma linha contendo um inteiro que representa o valor de saída (retorno) para a entrada N dada, ou um asterisco (*) no caso de o programa nunca terminar.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`9 6`
`IFEQ R0,0`
`RET 1`
`ENDIF`
`MOV R1,R0`
`SUB R1,1`
`CALL R1`
`MOV R2,R9`
`MUL R2,R0`
`RET R2`
`2 123`
`CALL R0`
`RET R0`
`0 0`
	`720`
	`*`

Problema C
Países em Guerra

Nome do arquivo fonte: `paises.c`, `paises.cpp`, `paises.java` *ou* `paises.pas`

No ano 2050, após diversas tentativas da ONU de manter a paz no mundo, explode a terceira guerra mundial. Segredos industriais, comerciais e militares obrigaram todos os países a utilizar serviços de espionagem extremamente sofisticados, de forma que em cada cidade do mundo há ao menos um espião de cada país. Esses espiões precisam se comunicar com outros espiões, com informantes e mesmo com as suas centrais durante as suas ações. Infelizmente não existe uma forma segura de um espião se comunicar em um período de guerra, então as mensagens são sempre enviadas em código para que somente o destinatário consiga ler a mensagem e entender o seu significado.

Os espiões utilizam o único serviço que funciona no período de guerra, os correios. Cada cidade possui uma agência postal onde as cartas são enviadas. As cartas podem ser enviadas diretamente ao seu destino ou a outras agências postais, até que a carta chegue à agência postal da cidade de destino, se isso for possível.

Uma agência postal na cidade A pode enviar diretamente uma carta impressa para a agência postal da cidade B se houver um acordo de envio de cartas, que determina o tempo, em horas, que uma carta leva para chegar da cidade A à cidade B (e não necessariamente o contrário). Se não houver um acordo entre as agências A e B, a agência A pode tentar enviar a carta a quantas agências for necessário para que a carta chegue ao seu destino, se isso for possível.

Algumas agências são interligadas por meios eletrônicos de comunicação, como satélites e fibras ópticas. Antes da guerra, essas ligações atingiam todas as agências, fazendo com que uma carta fosse enviada de forma instantânea, mas durante o período de hostilidades cada país passou a controlar a comunicação eletrônica e uma agência somente pode enviar uma carta à outra agência por meio eletrônico (ou seja, instantaneamente) se ela estiver no mesmo país. Duas agências, A e B, estão no mesmo país se houver uma forma de uma carta impressa enviada de uma das agências ser entregue na outra agência.

O serviço de espionagem do seu país conseguiu obter o conteúdo de todos os acordos de envios de mensagens existentes no mundo e deseja descobrir o tempo mínimo para se enviar uma carta entre diversos pares de cidades. Você seria capaz de ajudá-lo?

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém dois inteiros separados por um espaço, N ($1 \leq N \leq 500$) e E ($0 \leq E \leq N^2$), indicando o número de cidades (numeradas de 1 a N) e de acordos de envio de mensagens, respectivamente. Seguem-se, então, E linhas, cada uma com três inteiros separados por espaços, X, Y e H ($1 \leq X, Y \leq N$, $1 \leq H \leq 1000$), indicando que existe um acordo para enviar uma carta impressa da cidade X à cidade Y, e que tal carta será entregue em H horas.

Em seguida, haverá uma linha com um inteiro K ($0 \leq K \leq 100$), o número de consultas. Finalmente, virão K linhas, cada uma representando uma consulta e contendo dois inteiros separados por um espaço, O e D ($1 \leq O, D \leq N$). Você deve determinar o tempo mínimo para se enviar uma carta da cidade O à cidade D.

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir K linhas na saída. A I-ésima linha deve conter um inteiro M, o tempo mínimo, em horas, para se enviar uma carta na I-ésima consulta. Se não houver meio de comunicação entre as cidades da consulta, você deve imprimir "Nao e possivel entregar a carta"(sem acentos).

Imprima uma linha em branco após cada caso de teste.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
4 5 1 2 5 2 1 10 3 4 8 4 3 7 2 3 6 5 1 2 1 3 1 4 4 3 4 1 3 3 1 2 10 2 3 1 3 2 1 3 1 3 3 1 3 2 0 0	0 6 6 0 Nao e possivel entregar a carta 10 Nao e possivel entregar a carta 0

Problema D
Energia × Tempo

Nome do arquivo fonte: `energia.c`, `energia.cpp`, `energia.java` *ou* `energia.pas`

Paulo trabalha para uma grande empresa chamada Ábaco Computadores e Manutenções (ACM). O seu trabalho é prover manutenção de computadores de clientes da ACM, localizados em diversas partes do país. Por esta razão, Paulo normalmente passa um bom número de horas por semana dentro de aviões. Obviamente, Paulo sempre carrega consigo o seu laptop, de forma que mesmo quando está viajando de avião pode executar muitas tarefas relacionadas a seu trabalho.

Como as baterias de laptops geralmente não duram muito, Paulo tem estudado alternativas para aumentar o tempo de duração da bateria durante vôos. Ele descobriu que processadores modernos podem operar a diversos níveis de freqüência, oferecendo um compromisso entre desempenho e consumo de energia. A idéia inicial de Paulo foi simplesmente configurar o seu laptop na freqüência mais baixa. No entanto, ele notou que isso não era muito útil, já que as tarefas executavam muito lentamente no laptop, e não haveria tempo de executar todas as tarefas, de forma que a energia restante na bateria seria inútil.

Paulo notou, entretanto, que a influência do nível freqüência no desempenho varia de aplicação para aplicação, dependendo se elas são limitadas por memória, CPU ou E/S. Adicionalmente, como processadores modernos permitem que o nível de freqüência seja alterado por software, Paulo planeja utilizar esse mecanismo para aumentar o tempo de uso da bateria de seu laptop, de forma ainda a manter um desempenho razoável. Para levar em consideração tanto a energia como o desempenho, Paulo decidiu usar uma métrica já bem conhecida, denominada *Produto Energia × Tempo* (mais conhecida pelo acrônimo em inglês, EDP, *Energy × Delay Product*).

Paulo tem uma lista de programas que devem ser executados seqüencialmente, e todas as informações sobre o tempo e a energia necessários para executar cada programa em cada nível de freqüência, além da informação de quanta energia é gasta para fazer o processador mudar de freqüência. No entanto, para testar sua nova idéia, Paulo ainda tem um problema: como a maioria dos administradores de sistema, ele não gosta de programar. Ele está pedindo a sua ajuda, já que você é um grande amigo e um expert em algoritmos e programação, para determinar o nível de freqüência em que cada um de seus programas deve ser executado de forma a minimizar o EDP total.

Entrada

A entrada contém vários casos de testes. A primeira linha de um caso de teste contém quatro inteiros F, P, E, e A, identificando respectivamente o número de níveis de freqüência suportados pelo processador do laptop de Paulo ($1 \leq F \leq 20$), o número de programas a serem executados *seqüencialmente* ($1 \leq P \leq 5000$), a energia necessária, em Joules, para trocar entre dois quaisquer níveis de freqüência ($1 \leq E \leq 100$) e o tempo (em ms) para trocar entre quaisquer dois níveis de freqüência ($1 \leq A \leq 100$). Os níveis de freqüência são identificados por inteiros de 1 a F, e os programas são identificados por inteiros de 1 a P.

As $P \times F$ linhas seguintes descrevem os programas, com F linhas para cada programa (as primeiras F linhas correspondem ao programa 1, as próximas F linhas correspondem ao programa 2, e assim por diante). A *f-ésima* linha correspondente ao programa p contém dois números $E_{p,f}$ e $A_{p,f}$, representando respectivamente a quantidade de energia (em Joules) e

tempo (em ms) para executar o programa p no nível de freqüência f ($1 \leq E_{p,f} \leq 1000$ e $1 \leq D_{p,f} \leq 1000$). No início de cada caso de teste o processador está no nível 1 de freqüência. O final da entrada é indicada por $F = P = E = A = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada, seu programa deve produzir uma linha na saída, contendo o EDP mínimo para executar seqüencialmente o conjunto de programas de 1 a P (ou seja, na ordem em que aparecem na entrada).

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
2 3 10 10 50 120 100 90 500 600 600 500 400 1000 500 700 3 3 2 5 7 10 8 5 15 4 12 4 11 5 12 4 7 10 8 5 15 4 0 0 0 0	656100 145

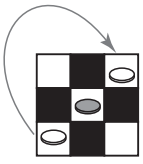
Problema E
MegaDamas

Nome do arquivo fonte: `damas.c`, `damas.cpp`, `damas.java` *ou* `damas.pas`

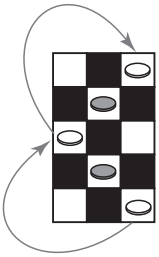
MegaDamas é um jogo de tabuleiro para dois jogadores, muito similar ao conhecido jogo de Damas. O tabuleiro é retangular, com N linhas e M colunas de pequenos quadrados arranjados em uma grade $N \times M$. Os pequenos quadrados são alternadamente coloridos com uma cor clara e uma cor escura, no padrão usual de um tabuleiro de damas. Os quadrados de cor escura são denominados "casas" (note no entanto que, por razões de visualização, os diagramas abaixo mostram casas como quadrados brancos).

No início do jogo, cada jogador tem um certo número de peças, posicionadas nas casas mais próximas da borda do tabuleiro que o jogador escolher (os jogadores escolhem bordas opostas). Durante o jogo, as peças só podem ocupar as casas do tabuleiro.

Um dos movimentos do jogo é "capturar" uma peça do oponente, saltando sobre ela, diagonalmente, para a casa adjacente além da peça, casa esta que deve estar vazia. A peça do oponente é então removida do tabuleiro. As três casas envolvidas na captura (a casa inicial de sua peça, a casa que contém a peça do oponente e a casa vazia, onde sua peça estará após a jogada) devem estar diagonalmente alinhadas e devem ser diagonalmente adjacentes, como no diagrama abaixo.



Captura simples



Captura múltipla

Em MegaDamas uma peça pode capturar peças do oponente saltando diagonalmente para a frente ou para trás (note que, na maioria das variações existentes do jogos de Damas, uma peça só pode capturar peças oponentes saltando para a frente). Você pode também efetuar uma captura múltipla, com uma peça apenas, saltando seguidamente para casas vazias sobre peças oponentes. Em uma captura múltipla, a sua peça pode mudar de direção, saltando primeiro em uma direção e depois em outra. Você pode capturar apenas uma peça a cada salto, mas pode capturar várias peças com saltos seguidos. Você não pode saltar sobre uma peça sua, e não pode saltar a mesma peça oponente mais de uma vez.

São dadas as dimensões do tabuleiro e uma descrição do estado corrente de um jogo. É a sua vez de jogar e você deve determinar o número máximo de peças do seu oponente que podem ser capturadas em um movimento de captura.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros N e M indicando respectivamente o número de linhas e o número de colunas do tabuleiro ($3 \leq N \leq 20$, $3 \leq M \leq 20$ e $N \times M \leq 200$). O quadrado mais à esquerda do tabuleiro na borda mais próxima ao jogador é uma casa. A segunda linha contém a descrição do estado do jogo.

Cada descrição consiste de $\lceil (N \times M)/2 \rceil$ inteiros, separados por um espaço, correspondendo às casas do tabuleiro, que são numeradas de 1 a $\lceil (N \times M)/2 \rceil$, da esquerda para a direita, da borda mais próxima ao jogador à borda mais próxima ao seu oponente. Na descrição do estado do jogo, '0' representa uma casa vazia, '1' representa uma casa com uma de suas peças, e '2' representa uma casa com uma peça de seu oponente. Há no máximo $\lfloor (N \times M)/4 \rfloor$ peças de cada jogador no tabuleiro. O final da entrada é indicado por $N = M = 0$.

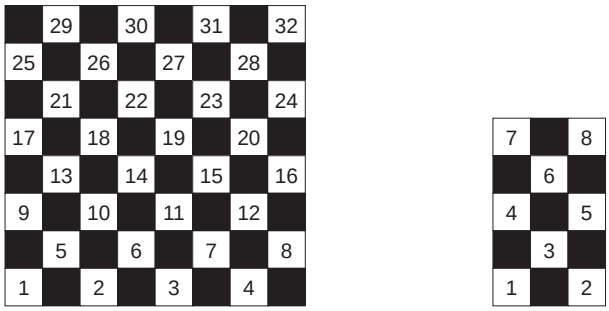


Figura 1: Numeração das casas em (a) tabuleiro de dimensões 8×8 e em (b) tabuleiro de dimensões 5×3.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada, seu programa deve produzir uma única linha na saída, contendo um inteiro indicando o maior número de peças de seu oponente que podem ser capturadas em uma jogada.

A saída deve ser escrita na saída padrão.

Exemplo de entrada

```
3 3
2 1 2 0 1
5 3
1 0 2 1 0 2 0 0
8 8
2 2 2 2 0 0 0 0 2 2 2 2 0 0 0 0 2 2 2 2 0 0 0 0 2 2 2 2 0 1 0 0
0 0
```

Saída para o exemplo de entrada

```
1
2
7
```

Problema F
Copa do Mundo

Nome do arquivo fonte: `copa.c`, `copa.cpp`, `copa.java` *ou* `copa.pas`

Uma Copa do Mundo de futebol de botões está sendo realizada com times de todo o mundo. A classificação é baseada no número de pontos ganhos pelos times, e a distribuição de pontos é feita da forma usual. Ou seja, quando um time ganha um jogo, ele recebe 3 pontos; se o jogo termina empatado, ambos os times recebem 1 ponto; e o perdedor não recebe nenhum ponto.

Dada a classificação atual dos times e o número de times participantes na Copa do Mundo, sua tarefa é de determinar quantos jogos terminaram empatados até o momento.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém dois inteiros T e N, indicando respectivamente o número de times participantes ($2 \leq T \leq 200$) e o número de partidas jogadas ($0 \leq N \leq 10000$). Cada uma das T linhas seguintes contém o nome de um time (uma cadeia de máximo 10 letras e dígitos), seguido de um espaço em branco, seguido do número de pontos que o time obteve até o momento. O final da entrada é indicado por $T = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada um dos casos de teste seu programa deve imprimir uma única linha contendo um número inteiro, representando a quantidade de jogos que terminaram empatados até o momento.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`3 3`
`Brasil 3`
`Australia 3`
`Croacia 3`
`3 3`
`Brasil 5`
`Japao 1`
`Australia 1`
`0 0`
	`0`
	`2`

Problema G
Rota Crítica

Nome do arquivo fonte: `rota.c`, `rota.cpp`, `rota.java` *ou* `rota.pas`

Uma tragédia aconteceu recentemente em sua cidade. Um paciente em condição crítica, que necessitava tratamento urgente, morreu enquanto era transportado para um grande hospital da capital do estado. O que ocorreu foi que a ambulância ficou presa no trânsito, devido a uma rocha que deslizou na estrada. A população reclamou com o governador, que agora deseja evitar acontecimentos similares no futuro. Infelizmente, deslizamentos de rochas são muito comuns nesse estado, com muitas montanhas e serras. Assim, para minimizar o número de tragédias devidas a deslizamentos de rochas e outros imprevistos, o governador decidiu criar rotas alternativas entre cada cidade do estado e a capital. Para isso, é necessário inicialmente identificar quais segmentos de estradas são atualmente *críticos*, isto é, se bloqueados causam que não haja caminho possível entre alguma cidade e a capital. Um segmento de estrada é um trecho de estrada que liga duas cidades distintas.

Sua tarefa é escrever um programa para identificar esses segmentos críticos de estradas.

Entrada

A entrada é composta de vários casos de testes. A primeira linha de um caso de teste contém dois inteiros N e M que indicam respectivamente o número de cidades ($2 \leq N \leq 100$) e o número de segmentos de estrada ($1 \leq M \leq 10000$). Cada uma das N linhas seguintes contém o nome de uma cidade (apenas letras minúsculas e maiúsculas, comprimento máximo de 20 caracteres). A primeira dessas cidades é a capital do estado. Cada uma das M linhas seguintes descreve um segmento de estrada, contendo um par de nomes de cidades separados por um espaço em branco. Note que, como as montanhas causam dificuldade na construção de estradas, muitos segmentos de estrada são de mão única. Um segmento com duas mãos é representado por dois trechos de mão única. Você deve supor que existe ao menos um caminho de cada cidade para a capital. O final da entrada é indicado por $N = M = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste seu programa deve listar os segmentos críticos, com um segmento crítico por linha. Cada segmento crítico deve ser representado por dois nomes de cidades separados por um espaço em branco. Os segmentos críticos de estrada devem ser listados na mesma ordem em que aparecem na entrada; para cada segmento, as cidades devem ser listadas na mesma ordem em que aparecem na entrada. Se não existir nenhum segmento crítico seu programa deve imprimir uma linha contendo apenas a palavra "`Nenhuma`". Imprima uma linha em branco após cada caso de teste.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
6 10 PortoAlegre Gramado Canela NovoHamburgo Pelotas RioGrande Canela Gramado Canela NovoHamburgo Gramado NovoHamburgo NovoHamburgo PortoAlegre PortoAlegre NovoHamburgo RioGrande Pelotas Pelotas PortoAlegre PortoAlegre Pelotas Pelotas RioGrande NovoHamburgo Canela 3 5 Sacramento SanFrancisco SantaClara SanFrancisco Sacramento Sacramento SantaClara SantaClara SanFrancisco SanFrancisco Sacramento Sacramento SanFrancisco 3 4 Recife Olinda Paulista Olinda Recife Paulista Recife Olinda Paulista Paulista Olinda 0 0	Gramado NovoHamburgo NovoHamburgo PortoAlegre RioGrande Pelotas Pelotas PortoAlegre SantaClara SanFrancisco Nenhuma

Problema H
Amigos ou Inimigos?

Nome do arquivo fonte: `amigos.c`, `amigos.cpp`, `amigos.java` *ou* `amigos.pas`

Um determinado exército numa certa fronteira decidiu enumerar as coordenadas em sua volta de maneira a tornar difícil para o inimigo saber a quais posições eles estão se referindo no caso de o sinal de rádio usado para comunicação ser interceptado. O processo de enumeração escolhido foi o seguinte: primeiro decide-se onde ficam os eixos `x` e `y`; a seguir, define-se uma equação linear que descreva a posição da fronteira em relação aos eixos (sim, ela é uma linha reta); finalmente, enumeram-se todos os pontos do plano cartesiano que não fazem parte da fronteira, sendo o número `0` atribuído à coordenada `(0,0)` e daí em diante atribuindo-se os números para as coordenadas inteiras seguindo uma espiral de sentido horário, sempre pulando os pontos que caem em cima da fronteira (veja a Figura 1). Caso o ponto `(0,0)` caia em cima da fronteira, o número `0` é atribuído ao primeiro ponto que não faça parte da fronteira seguindo a ordem especificada.

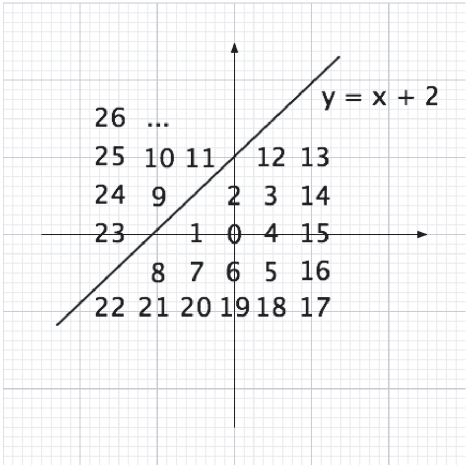


Figura 2: Enumeração dos pontos das coordenadas inteiras

De fato o inimigo não tem como saber a qual posição o exército se refere, a não ser que o inimigo saiba o sistema usado para enumerar os pontos. Tal esquema, porém, complicou a vida do exército, uma vez que é difícil determinar se dois pontos quaisquer estão no mesmo lado da fronteira ou em lados opostos. É aí que eles precisam da sua ajuda.

Entrada

A entrada contém vários casos de teste. A primeira linha da entrada contém um inteiro N ($1 \leq N \leq 100$) que representa a quantidade de casos de teste. Seguem-se os N casos de teste. A primeira linha de cada caso de teste contém dois inteiros a e b ($-5 \leq a \leq 5$ e $-10 \leq b \leq 10$), que descrevem a equação da fronteira: $y = ax + b$. A segunda linha de cada caso de teste contém um inteiro K, indicando a quantidade de consultas que se seguem ($1 \leq K \leq 1000$). Cada uma das K linhas seguintes descreve uma consulta, sendo composta por dois inteiros M e N representando as coordenadas enumeradas de dois pontos ($0 \leq M, N \leq 65535$).

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir $K + 1$ linhas. A primeira linha deve conter a identificação do caso de teste na forma `Caso X`, onde `X` deve ser substituído pelo número do caso (iniciando de `1`). As K seguintes devem conter os resultados das K consultas feitas no caso correspondente da entrada, na forma:

```
Mesmo lado da fronteira
```

ou

```
Lados opostos da fronteira
```

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
2 1 2 10 26 25 25 11 24 9 23 28 25 9 25 1 25 0 9 1 23 12 26 17 1 2 12 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12	Caso 1 Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Lados opostos da fronteira Lados opostos da fronteira Lados opostos da fronteira Lados opostos da fronteira Lados opostos da fronteira Caso 2 Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Lados opostos da fronteira Mesmo lado da fronteira Mesmo lado da fronteira Lados opostos da fronteira