Young

1ª Maratona de Programação dos Alunos da UFPR – 2011/1

Sevidor BOCA:

`http://maratona.c3sl.ufpr.br/boca/`

Organizadores:

Bruno César Ribas, André Luiz Guedes, Eduardo Augusto Ribas

Revisores:

Roberto Hexsel, Marcos Castilho

Lembretes:

- 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.

Prova Young

1ª Maratona de Programação dos Alunos da UFPR

28 de junho de 2011

Sumário

1 Problema A: Grade Horária

`Arquivo: grade.[c|cpp|java|pas]`

Professor Toddy é um simpático e calmo coordenador do curso *Bacharelado em Chef de Cozinha* (BCC) que, como em todo fim de semestre, começa a montar a grade horária das disciplinas para o próximo semestre. Professor Toddy é famoso por sempre tentar colocar as disciplinas em horários que os alunos mais desejam, e isso acontece principalmente para as disciplinas que são optativas. Para entender melhor as demandas dos alunos o professor passa um questionário. No questionário, o professor lista todas as disciplinas optativas que poderão ser ofertadas (e seus respectivos horários), oferecendo uma coluna para o aluno selecionar a optativa que deseja cursar no próximo semestre ou para vetá-la. É permitido fazer apenas duas escolhas no questionário, ou seja, cada aluno pode selecionar uma optativa e vetar outra, vetar duas optativas ou selecionar duas optativas. O professor Toddy garante que todos os alunos terão pelo menos um de seus desejos atendidos.

Antigamente ele mesmo dava conta de montar a grade de optativas e atender o que prometia, mas com o crescimento do curso e a grande quantidade de alunos tem se tornado impossível. Assim, ele resolveu contratar você para fazer um programa que recebe os pedidos dos alunos e responde se é possível montar uma grade de optativas para o próximo semestre.

Entrada

A entrada é composta de diversas instâncias. Cada instância começa com um inteiro n ($1 \leq n \leq 1000$), indicando a quantidade de questionários recebidos pelo professor Toddy. Cada uma das próximas n linhas contém dois nomes de disciplina indicando a preferencia de cada aluno. Um nome de disciplina é uma sequencia de letras `[a-z]` com no máximo 20 letras. Quando o nome de uma disciplina é iniciado por "`!`" significa que o aluno deseja vetar a disciplina, caso contrário ele deseja selecionar.

O final da entrada é dado por $n = 0$.

Saída

Para cada instância, você deverá imprimir um identificador `Instancia k`, onde k é o número da instância atual. Na linha seguinte você deve imprimir `sim` se for possível atender pelo menos um desejo de cada aluno e `nao` caso contrário.

Após cada instância, seu programa deve imprimir uma linha em branco.

Exemplo

Entrada

```
2
!tap !itc
mc !tap
4
mc bd
!mc !bd
mc !bd
!mc bd
```

Saída

```
Instancia 1
sim

Instancia 2
nao
```

Curiosidades

tap - Tópicos em Arroz Parboilizado

mc - Metodologia Culinária

bd - Banco de Doces

itc - Introdução às Técnicas de Churrasco

2 Problema B: Fechem as Portas

`Arquivo: portas.[c|cpp|java|pas]`

Madame Beauvoir possui uma mansão onde ela recebe todos os seus descendentes (netos e bisnetos) durante as férias. Sua mansão possui exatamente N quartos (cada quarto é numerado de 1 a N), onde N é também a quantidade de netos e bisnetos (cada descendente é também numerado de 1 a N).

Como toda criança, os descendentes de Mme. Beauvoir são bastante travessos. Todo dia é a mesma confusão: eles acordam de manhã cedo antes dela e se encontram no grande jardim. Cada descendente, um de cada vez, entra na mansão e troca o estado das portas dos quartos cujos números são múltiplos do seu identificador. Trocar o estado de uma porta significa fechar uma porta que estava aberta ou abrir uma porta que estava fechada. Por exemplo, o descendente cujo identificador é igual a 15 vai trocar o estado das portas 15, 30, 45, etc.

Considerando que todas as portas estão inicialmente fechadas (todos os descendentes fecham as portas antes de descer para o jardim) e que cada descendente entra exatamente uma vez na mansão (a confusão é tão grande que não sabemos em que ordem), quais portas estarão abertas após a entrada de todos os descendentes na mansão?

Entrada

A entrada contém vários casos de teste. Cada caso de teste consiste em uma linha que contém um inteiro N ($0 < N \leq 25000000$), indicando o número de portas e descendentes. 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 uma linha na saída, contendo a sequência crescente de números correspondente aos identificadores dos quartos cujas portas estarão abertas. Ao imprimir a sequência, deixe um espaço em branco entre dois elementos consecutivos.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Exemplo de saída
1	
2	1
3	1
4	1
0	1 4

3 Problema C: Ajude um Candidato ao Doutorado!

`Arquivo: doutorado.[c|cpp|java|pas]`

João o Gênio esqueceu como somar dois números enquanto pesquisava em seu doutorado. E agora ele tem uma longa lista de problemas de adição que precisa resolver, além daqueles inerentes à ciência da computação! Você pode ajudá-lo?

Em sua lista atual, João o Gênio tem dois tipos de problemas: adição na forma "$a + b$" e o recorrente problema "P=NP". João o Gênio é uma pessoa muito distraída, então ele deve resolver este último problema diversas vezes, pois continua esquecendo a solução. Além disto, ele gostaria de resolver sozinho, então você deve ignorá-los.

Especificação da Entrada

A primeira linha de entrada consiste em um único inteiro N ($1 \leq N \leq 1000$), denotando o número de casos de teste. Então seguem N linhas com "P=NP" ou um problema de adição na forma "$a + b$", onde $a, b \in [0, 1000]$ são inteiros.

Especificação da Saída

Exiba o resultado de cada adição. Para linhas contendo "P=NP", apresente "`skipped`".

Exemplo de entrada

```
4
2+2
1+2
P=NP
0+0
```

Exemplo de saída

```
4
3
skipped
0
```

4 Problema D: Detectando Colisões!

`Arquivo: colisoes.[c|cpp|java|pas]`

Detecção de colisão é uma das operações mais comuns (e importantes) em jogos eletrônicos. O objetivo, basicamente, é verificar se dois objetos quaisquer colidiram, ou seja, se a interseção entre eles é diferente de vazio. Isso pode ser usado para saber se duas naves colidiram, se um monstro bateu numa parede, se um personagem pegou um item, etc.

Para facilitar as coisas, muitas vezes os objetos são aproximados por figuras geométricas simples (esferas, paralelepípedos, triângulos etc). Neste problema, os objetos são aproximados por retângulos num plano 2D.

Tarefa

Escreva um programa que, dados dois retângulos, determine se eles se interceptam ou não.

Entrada

A entrada contém vários conjuntos de testes, que deve ser lido do *dispositivo de entrada padrão*. A primeira linha contém um número N ($1 \leq N \leq 1000$) que indica quantos casos de teste deverão ser processados. Cada caso de teste contém duas linhas. Cada linha contém quatro inteiros (x_0, y_0, x_1, y_1, sendo $0 \leq x_0 < x_1 \leq 1000000$ e $0 \leq y_0 < y_1 \leq 1000000$) separados por um espaço em branco representando um retângulo. Os lados do retângulo são sempre paralelos aos eixos x e y.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha para cada caso de teste, contendo o número 0 (zero) caso não haja interseção ou o número 1 (um) caso haja.

Entrada

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

Saída

```
1
1
0
```

5 Problema E: Labirinto!

`Arquivo: labirinto.[c|cpp|java|pas]`

Um amigo seu está muito empolgado com um novo joguinho que baixou em seu celular. O jogo consiste em uma espécie de labirinto que pode ser representado por um quadriculado de células quadradas com N linhas e M colunas. Cada célula do labirinto contém uma plataforma que está a uma determinada altura do chão, que pode ser representada por um inteiro a que varia de 0 (a mais baixa) a 9 (a mais alta). Você inicia na célula $(1, 1)$ (canto superior esquerdo) e o objetivo é chegar na saída do labirinto que fica na célula (N, M) (canto inferior direito).

Para sair do labirinto, você deve fazer movimentos entre células adjacentes. O problema é que seu bonequinho não consegue pular muito alto, então se a célula destino estiver duas ou mais unidades acima da sua altura atual, você não consegue movê-lo. Mais especificamente, a cada turno você pode mover para uma das 4 células adjacentes (norte, sul, leste, oeste) *caso a altura da célula destino seja menor ou igual à altura da sua célula atual mais uma unidade*. Ou seja, se a altura da sua célula for A, você só pode mover a uma célula adjacente caso a altura dela seja menor ou igual a $A + 1$.

Para complicar um pouco mais o jogo, a cada turno, *após o jogador realizar sua ação*, cada célula aumenta em uma unidade sua altura, até o valor máximo 9. Caso a altura de uma determinada célula seja 9, ela passa a ser 0.

Note que, em um dado turno, o jogador não é obrigado a se mover, ele pode simplesmente esperar as plataformas subirem ou descerem. Além disso, repare que nem todas as células têm 4 vizinhos, uma vez que não é permitido ao jogador se mover para fora dos limites do labirinto.

Você, como bom programador que é, resolve escrever um programa que calcule a menor quantidade de turnos possível para chegar à saída de um dado labirinto.

Tarefa

Escreva um programa que, dado um labirinto, retorne a menor quantidade de turnos necessária para chegar à saída, de acordo com as restrições dadas.

Entrada

A entrada contém vários conjuntos de testes, que deve ser lido do *dispositivo de entrada padrão*. A primeira linha de um conjunto de teste contém dois inteiros N e M ($2 \leq N, M \leq 50$) separados por um espaço em branco, que representam, respectivamente, a quantidade de linhas e colunas do labirinto. As N linhas seguintes contêm, cada uma, M inteiros que representam a altura inicial (no turno 0) da respectiva plataforma. As alturas estão sempre entre 0 e 9 (inclusive). O final da entrada é indicado por $N = M = 0$.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha para cada caso de teste, contendo a menor quantidade de turnos possível para sair do labirinto.

Exemplo de entrada

```
4 3
0 0 0
0 0 0
0 0 0
0 0 0
3 3
1 2 3
4 5 6
7 8 9
3 5
1 3 1 1 1
1 3 1 3 1
1 1 1 3 1
0 0
```

Exemplo de Saída

```
5
12
10
```

6 Problema F: Difícil de Acreditar, mas Verdade!

`Arquivo: mas_verdade.[c|cpp|java|pas]`

A luta continua, na decisão sobre armazenar números iniciados por seus dígitos mais significativos ou menos significativos. Às vezes isto é chamado de "*Endian War*". O campo de batalha remonta aos períodos do início da Ciência da Computação. Joe Stoy, em seu (a propósito excelente) livro "*Denotational Semantics*", nos conta a seguinte estória:

> A decisão sobre em que direção os dígitos são dispostos é, naturalmente, matematicamente trivial. De fato, um computador britânico antigo apresentava números da direita para a esquerda (porque o cursor num osciloscópio funcionava da esquerda para a direita, mas em lógica serial os dígitos menos significativos eram processados antes). Turing costumava assombrar a audiência em suas palestras quando, quase por acidente, ele se utilizava desse sistema e escrevia coisas como `73+42=16`. A versão seguinte da máquina foi construída de forma mais convencional simplesmente trocando os fios de deflexão no eixo X: o que, no entanto, preocupava os engenheiros pois seus gráficos de onda apareciam invertidos. Este problema foi por sua vez resolvido através de uma pequena janela onde os engenheiros (que geralmente costumavam estar atrás do computador de qualquer forma) pudessem ver a tela do osciloscópio por trás. [C. Strachey - comunicação privada]

Você fará o papel da audiência e julgar a veracidade das equações de Turing.

Especificação da Entrada

A entrada contém vários casos de teste. Cada um especifica uma equação de Turing na forma $a + b = c$, onde a, b, c são números compostos por dígitos 0..9. Cada número pode conter no máximo 7 dígitos. Isto inclui número iniciados ou seguidos de zeros. A equação "0+0=0" encerra a entrada e precisa ser processada também. Não existem espaços em branco no meio das equações.

Especificação da Saída

Para cada caso de teste, imprima "True" ou "False" indicando se a equação for verdadeira ou falsa, respectivamente, de acordo com a interpretação de Turing (os números sendo lidos invertidos).

Exemplo de entrada

```
73+42=16
5+8=13
10+20=30
0001000+000200=00030
1234+5=1239
1+0=0
7000+8000=51
0+0=0
```

Exemplo de saída

```
True
False
True
True
False
False
True
True
```

7 Problema G: Quem vai ser reprovado

`Arquivo: reprovado.[c|cpp|java|pas]`

Prof. Castellone da Universidade de Toulouse está muito preocupado com a queda do nível de atenção de seus estudantes. Ele já tentou várias técnicas mundialmente conhecidas para incentivar os alunos a prestar atenção nas suas aulas e fazer as tarefas que ele passa para a turma: deu nota para os alunos mais participativos, ofereceu chocolates aos alunos, levou seu karaokê e cantava nas aulas etc. Como tais medidas não levaram a uma melhora no comparecimento às aulas (a ideia do karaokê, inclusive, mostrou-se bastante infeliz... na segunda aula com karaokê a turma reduziu-se a um aluno – que tinha problemas auditivos) ele teve uma brilhante ideia: faria uma competição entre os alunos.

Prof. Castellone passou um conjunto de problemas aos alunos, e deu um mês para que eles os resolvessem. No final do mês os alunos mandaram o número de problemas resolvidos corretamente. A promessa do brilhante didata era reprovar sumariamente o último colocado da competição. Os alunos seriam ordenados conforme o número de problemas resolvidos, com empates resolvidos de acordo com a ordem alfabética dos nomes (não há homônimos na turma). Isso fez com que alunos com nomes iniciados nas últimas letras do alfabeto se esforçassem muito nas tarefas, e não compartilhassem suas soluções com colegas (especialmente aqueles cujos nomes começassem com letras anteriores). Sua tarefa neste problema é escrever um programa que lê os resultados dos alunos do Prof. Castellone e imprime o nome do infeliz reprovado.

Entrada

A entrada é composta de diversas instâncias. A primeira linha de cada instância consiste em um inteiro n ($1 \leq n \leq 100$) indicando o número de alunos na competição. Cada uma das n linhas seguintes contém o nome do aluno e o número de problemas resolvidos por ele. O nome consiste em uma sequência de letras `[a-z]` com no máximo 20 letras e cada aluno resolve entre 0 a 10 problemas.

A entrada termina com final de arquivo.

Saída

Para cada instância, você deverá imprimir um identificador `Instancia k`, onde k é o número da instância atual. Na linha seguinte imprima o nome do infeliz reprovado.

Após cada instância imprima uma linha em branco.

Exemplo de Entrada

```
4
cardonha 9
infelizreprovado 3
marcel 9
infelizaprovado 3
```

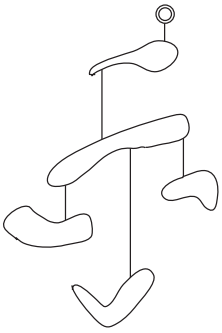
Exemplo de saída

```
Instancia 1
infelizreprovado
```

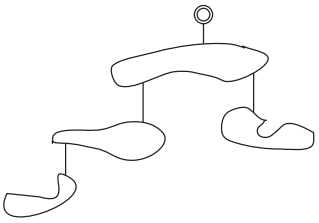
8 Problema H: Móbile

`Arquivo: mobile.[c|cpp|java|pas]`

Móbiles são objetos muito populares hoje em dia, sendo encontrados até em berços, para diversão de bebês, mas foram concebidos há muito tempo (em 1931) pelo então jovem artista americano Alexander Calder como esculturas em movimento. Um móbile é uma estrutura composta de peças unidas por fios. O móbile é preso por um fio a uma argola pela qual ele é suspenso, permitindo que a estrutura movimente-se livremente. A argola é presa a uma única peça, chamada de peça-raiz do móbile. A peça-raiz pode ter zero ou mais sub-móbiles pendurados nela, cada sub-móbile sendo composto por uma peça-raiz na qual por sua podem estar pendurados zero ou mais sub-móbiles, e assim sucessivamente. Abaixo podemos ver dois exemplos de móbiles:



(a)



(b)

Victor é dono de uma fábrica de móbiles que emprega centenas de artesãos. Cada móbile produzido na fábrica é confeccionado por um artesão, que cria móbiles de acordo com o seu gosto pessoal, utilizando peças de formatos distintos. Entretanto, Victor tem notado que nem todos os seus artesões possuem a mesma habilidade artística, de forma que às vezes o móbile produzido nem sempre é bem balanceado, segundo a sua concepção. Para Victor, um móbile é bem balanceada se, para cada peça, todos os sub-móbiles pendurados nela são compostos pelo mesmo número de peças. O número de peças de um sub-móbile é determinado contando-se o número de peças que o compõem, incluindo a sua peça-raiz. Note que cada peça do móbile, exceto a peça-raiz, é pendurada em exatamente uma outra peça.

Por exemplo, o móbile da figura (a) acima é um móbile bem balanceado: a peça-raiz possui um único sub-móbile que por sua vez possui três sub-móbiles, todos com o mesmo número de peças (uma única). Já o móbile da figura (b) é um móbile mal balanceado: a peça-raiz possui dois sub-móbiles, um com o total de duas peças e outro com o total de uma peça.

Tarefa

Dada a descrição de um móbile, você deve escrever um programa para determinar se o móbile está bem balanceado ou não.

Entrada

A entrada contém vários conjuntos de testes, que deve ser lido da entrada padrão. A primeira linha da entrada contém um inteiro N que indica o número de peças utilizadas no móbile ($1 \leq N \leq 10000$). As peças são identificadas por inteiros de 1 a N. Cada uma das N

linhas seguintes contém dois números inteiros I e J, indicando que a peça de número I está pendurada na peça de número J (a peça raiz está pendurada na argola, que é identificada pelo o número 0). O final da entrada é indicado por $N = 0$.

Saída

Seu programa deve imprimir, na saída padrão, uma única linha para cada caso de teste, contendo a palavra **bem** se o móbile estiver bem balanceado ou **mal** caso esteja mal balanceado. A palavra deve ser escrita com todas as letras em minúsculas.

Exemplo de entrada

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

Exemplo de saída

```
bem
bem
mal
```