8ª Maratona de Programação

Facens 2012

CADERNO DE PROBLEMAS

PROBLEMA A: iLOGICA

Arquivo: `ilogica.[cpp/c/java]`

Cor: rosa

Limite de tempo: 2s

Descrição do problema

Uma empresa de eletrônicos chamada iLÓGICA está desenvolvendo um simulador para seu modelo de DLP (dispositivo lógico programável). Um DLP pode ser configurado para armazenar um circuito lógico que, dado um determinado número de entradas lógicas (verdadeiro ou falso), forneça uma saída lógica.

O simulador realizará a mesma tarefa. Por isso, ele deve interpretar o código usado para programar o DLP e dar a mesma resposta. Você, um engenheiro conceituado da FACENS, foi contratado para programar a parte do simulador que interpretará expressões lógicas usado no código do simulador. As expressões lógicas são compostas de algumas operações lógicas conhecidas, a saber:

- Operação "AND": para duas entradas lógicas verdadeiras, essa operação resulta em uma saída verdadeira. Em qualquer outro caso, a operação resulta em uma saída falsa.
- Operação "OR": para qualquer entrada lógica verdadeira, essa operação resulta em uma saída verdadeira, Em caso de duas entradas lógicas falsas, a saída será falsa.
- Operação "NOT": para uma entrada verdadeira, a saída será falta, e, para uma entrada falsa, a saída será verdadeira.

As entradas e saídas verdadeiras podem ser representadas pelo número **1**. Já as entradas e saídas falsas podem ser representadas pelo número **0**. Nesse caso, temos as tabelas para as entradas e saídas em cada operação:

Operação AND

Entradas		Saída
0	0	0
0	1	0
1	0	0
1	1	1

Operação OR

Entradas		Saída
0	0	0
0	1	1
1	0	1
1	1	1

Operação NOT

Entrada	Saída
0	1
1	0

A expressão lógica que o simulador vai interpretar segue o seguinte modelo:

A and B – operação AND tendo como entrada as variáveia A e B

A or B – operação OR tendo como entrada as variáveis A e B

not A – operação NOT tendo como entrada a variável A.

Uma expressão poder ser uma combinação dessas operações, como nos seguintes exemplos:

```
A and B

not B or A

C and A or D and not B

not B or not A and B or C and not D or E or F

A or A
```

Para interpretar uma expressão, é necessário saber a precedência das operações. A operação **"not"** tem maior precedência, seguida da operação **"and"** e por último, a operação **"or"**. Ou seja, ao analisar a expressão, primeiro resolvem-se as operações **"not"**, na sequência as operações **"and"** e, no final, as operações **"or"**.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste possui 2 linhas. A primeira linha é a expressão a ser processada (com um máximo de 200 caracteres). A segunda linha é o valor de cada uma das variáveis de entrada, separados por espaço, sempre iniciando em "A" e seguindo na sequência, dependendo do número de variáveis (B, C, D, E e F). O número máximo de variáveis é 6. O valor -1 na primeira linha de uma entrada indica o final dos casos de teste.

Saída

Para cada caso de teste deverá ser apresentado um valor, 0 ou 1.

Exemplos de testes

Entrada	Saída
A and B 0 1 not A or B and C 0 0 0 A and B or C and not D 0 0 1 1 A and B or C and not D 1 1 0 0 not A and not B and not A A or A 0 -1	0 1 0 1 0

PROBLEMA B: SHOW ME THE CODE

Arquivo: `thecode.[cpp/c/java]`

Cor: amarelo

Limite de tempo: 1s

Descrição do problema

Após a repercussão de alguns atentados virtuais ao redor do globo, segurança digital está em alta no mercado. Aproveitando a oportunidade, você arrumou um emprego em uma empresa especializada em codificação e decodificação de mensagens.

Como sua primeira tarefa, uma série de códigos e sua decodificação numérica lhe foram entregues. Você precisa identificar qual é o padrão de codificação e tentar quebrá-lo, escrevendo um programa que converta as mensagens codificadas no seu código numérico. Sua única pista é que o criador do código adora utilizar as formas dos símbolos na codificação. A única dica que lhe foi dada é que o criador dos códigos gosta muito de utilizar formas de símbolos para criar suas criptografias.

Entrada

A entrada é composta de vários códigos. Cada código C é um número entre zero e 99.999. O código -1 indica o final dos códigos.

Saída

Para cada caso de teste deverá ser apresentado um valor numérico que corresponde a decodificação do código.

Exemplos de testes

Entrada	**Saída**
30	2
800	0
1000	3
552	6
44	6
999	3
81014	9
87856	5
81603	6
70545	9
62598	6
15943	11
47214	13
36159	9
78300	4
9272	7
73780	6
86521	8
80797	5
1307	7
27514	12
-1	

PROBLEMA C: PROMESSAS DE CAMPANHA

Arquivo: `avenidas.[cpp/c/java]`

Cor: roxo

Limite de tempo: 1s

Descrição do problema

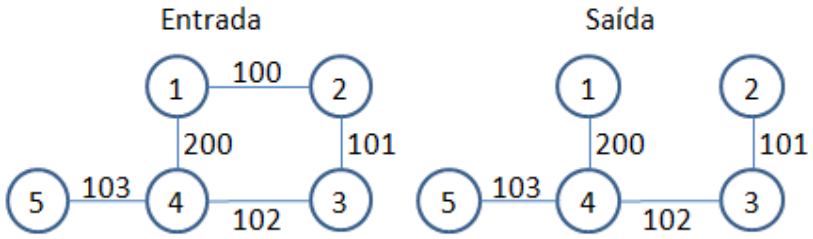
Sendo um ativista político em um ano de eleições, você decidiu que era hora de ajudar o seu partido. Como primeira tarefa o candidato a prefeito pediu que você construísse um programa que o ajude a determinar quais avenidas ele construirá caso seja eleito. Para tal tarefa seu programa deve receber uma lista de possíveis avenidas (essa lista contém os bairros interligados e a qualidade das avenidas) e, a partir dela, deve definir quais delas devem ser construídas segundo as seguintes regras:

- O menor número de avenidas deve ser construído;
- A qualidade das avenidas deve ser maximizada;
- Deve ser possível sair de um bairro e chegar a qualquer outro utilizando apenas as avenidas construídas;

Para ajudá-lo a resolver o problema seguem algumas características interessantes das avenidas contidas na lista:

- Duas avenidas nunca possuem a mesma qualidade;
- Todas as avenidas são de mão dupla;
- Pode existir mais de uma avenida ligando dois bairros;

A imagem abaixo ilustra o primeiro caso de teste.



Entrada

A entrada é constituída de vários casos de teste. Cada caso de teste começa com o valor N, que corresponde a quantidade de possíveis avenidas. Em seguida, são informadas N avenidas no formato A B1 B2 Q, que representam o identificador da avenida, o primeiro bairro, o segundo bairro e a qualidade da avenida, respectivamente. O final de entrada é indicado por N = 0. As restrições são:

1 <= N <= 100

1 <= A <= 500

1 <= B1 <= 500

1 <= B2 <= 500

1 <= Q <= 100

Saída

A saída deverá ser composta por uma linha contendo a lista de avenidas que serão construídas separadas por vírgula e ordenadas pelo identificador.

Exemplos de testes

Entrada	**Saída**
5 2 2 3 101 4 4 5 103 1 1 2 100 3 3 4 102 5 1 4 200 3 1 1 2 50 3 1 3 1 2 2 3 101 0	2,3,4,5 1,2

PROBLEMA D: LORD OF THE TOKENS

Arquivo: `lingreg.[cpp/c/java]`

Cor: branco

Limite de tempo: 1s

Descrição do problema

Na teoria de compiladores duas etapas são necessárias para se construir um compilador. Essas etapas são chamadas de análise e síntese. Um compilador é um programa que lê uma entrada escrita em uma linguagem (código fonte) e o traduz num programa equivalente numa outra linguagem (programa alvo).

Conceitualmente, um compilador opera em fases, cada uma das quais transforma o programa fonte de uma representação para outra.

Sua equipe foi contratada para desenvolver um programa que utilizando os conceitos de compiladores (considere apenas a fase léxica e sintática) construa um programa que receba como entrada registros no seguinte formato (Nome da faculdade, RA e disciplinas) e diga se foram ou não reconhecidos.

Na fase da análise léxica são identificados os tokens reconhecidos pela linguagem definida e na análise sintática são identificados se esses tokens estão adequadamente posicionados. Um token é uma sequência justaposta de caracteres.

Entrada

A entrada é constituída de vários casos de teste. Cada caso possui o nome da faculdade (apenas letras maiúsculas), o RA (apenas números) e o nome das disciplinas (cada disciplina contém apenas 2 letras maiúsculas e 3 algarismos). O programa deverá ser encerrado quando a linha se iniciar com $.

Saída

A saída será composta pela palavra reconhecida ou nao reconhecida.

Exemplos de testes

Entrada	**Saída**
FACENS 123456 XY123 WR456 A 12 AB123 FaCENS 8 AB123 $	reconhecida reconhecida nao reconhecida

PROBLEMA E: MOLECULAS

Arquivo: `moleculas.[cpp/c/java]`

Cor: verde

Limite de tempo: 1s

Descrição do problema

O laboratório de pesquisas modernas, LPM, está trabalhando com experimentos moleculares altamente avançados. O laboratório está desenvolvendo uma nova técnica para realizar dobras em sequências de moléculas, gerando um sequenciamento ligado entre si.

O procedimento em si depende de uma reorganização estrutural da sequência, de forma que a dobra possa ser realizada. Como o procedimento só pode ser realizado em algumas circunstâncias, o LPM possui muitos problemas já que normalmente as sequências podem possuir um número muito grande de moléculas.

O exemplo abaixo mostra quando uma dobra pode ser realizada e como é o procedimento:

Sequência de moléculas originais

B	C	D	C	B	A	D	A

Molécula reorganizada para realizar a dobra

B	C	D	A	A	D	C	B

Molécula dobrada

B	C	D	A
B	C	D	A

Para auxiliar no desenvolvimento da técnica, o LPM está solicitando a sua ajuda para identificar os sequenciamentos de moléculas que podem ser dobrados.

Entrada

A entrada do problema é composta por uma cadeia de caracteres, onde cada caractere corresponde a uma molécula diferente. A cadeia é composta por N caracteres, onde 0<N<5000. A entrada se encerra com uma sequência vazia.

Saída

A saída é composta por um texto indicando se é possível realizar a dobra ou não.

Exemplos de testes

Entrada	**Saída**
BCDCBADA ABCDPQRABC ORRPOXTPT NHJIJJHN	SIM NAO SIM NAO

PROBLEMA F: QUANTO MAIS MELHOR

Arquivo: `qmm.[cpp/c/java]`

Cor: preto

Limite de tempo: 1s

Descrição do problema

O fotografo Chico Treva é nacionalmente conhecido pelo perfeccionismo nas imagens que publica. Para chegar a essa "perfeição", ele normalmente tira 30 ou 40 fotos do mesmo cenário para escolher a melhor. Porém em um dos seus últimos trabalhos ele, como sempre fez, fotografou diversas vezes a cena, mas devido a um problema na câmera, todas as imagens saíram com ruídos aleatórios.

Como o prazo para entrega do material estava apertado, não seria possível refazer todas as fotos. Dessa forma, ele recorreu a computação para tentar resolver o problema.

A solução encontrada foi utilizar a sequência de imagens de um mesmo objeto para calcular uma imagem média e eliminar o ruído e tomar a imagem mais nítida, uma das características dos trabalhos do Chico.

Entrada

A entrada é composta de vários casos de testes, sendo que a primeira linha deve conter as dimensões das imagens (1 <= X <= 1000 e 1 <= Y <= 1000), na linha seguinte o número de imagens que serão utilizadas para calcular a imagem média e por fim as imagens, representadas por uma matriz de caracteres ASCII. Os casos de testes terminam quando as dimensões da matriz forem -1 -1.

Saída

Para cada caso de teste, a saída deverá conter uma matriz do mesmo tamanho das imagens de entrada com a imagem média.

Exemplos de testes

Entrada	**Saída**
3 3 4 AAA ABA AAA AAB AAA AAA AAA AAA ABA AAA AAA BAA 4 4 3 **** ***+ **** +*** **** ***+ ***+ **** **** ***+ +*** -1 -1	SIM NAO SIM NAO

PROBLEMA G: JUROS POR TUDO QUE É MAIS SAGRADO

Arquivo: `juros.[cpp/c/java]`

Cor: laranja

Limite de tempo: 1s

Descrição do problema

O novo presidente da empresa onde você trabalha pediu que uma nova informação fosse adicionada ao relatório mensal de despesas. Atualmente, o relatório exibe, entre outras coisas, o valor dos juros mensal do empréstimo realizado para expandir a empresa. Agora, o presidente deseja saber o valor dos juros diário de tal empréstimo. Para calcular o valor dos juros diário você deve considerar a quantidade de dias do mês do relatório; considere que fevereiro sempre possui 28 dias. Dica: o valor dos juros diário deve ser calculado como juros composto.

Entrada

A entrada é constituída de vários casos de teste. Cada caso de teste possui dois valores J e M que correspondem ao valor dos juros mensais e o mês do relatório, respectivamente. O final de entrada é indicado por J = M = 0.

Restrições:

1.00 <= J <= 9.99
1 <= M <= 12

Saída

A saída deverá ser composta por uma linha contendo o valor dos juros diário do empréstimo realizado com precisão de dez casas decimais.

Exemplos de testes

Entrada	**Saída**
7.53 2 1.08 3 5.67 12 0 0	0.2596210490% 0.0346579288% 0.1780642964%

PROBLEMA H: PLACAR RETURNS

Arquivo: `placar.[cpp/c/java]`

Cor: azul

Limite de tempo: 5s

Descrição do problema

A Liga Nacional de Basquete (LNB) foi lançada em Dezembro de 2008, reunindo as principais lideranças e os mais representativos clubes do basquete brasileiro, com o objetivo de reconduzir o esporte ao posto de segundo mais popular do Brasil, atrás apenas do futebol. A LNB conta com 18 clubes associados, que participam do Novo Basquete Brasil (NBB), campeonato brasileiro masculino adulto, organizado pelos clubes.

Devido à intenção de recuperar a popularidade desse esporte, são feitos investimentos e pesquisas para melhorar a sistemática das partidas, o desempenho dos jogadores e a atuação dos árbitros. Dentro desse contexto, uma confederação de basquete contratou a sua equipe para desenvolver um programa que, a partir do placar final de um jogo e do número de lances livres convertidos por cada equipe, determine quantas possíveis sequências de cestas existem para chegar ao placar final.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste possui quatro valores, sendo os dois primeiros os totais de pontos marcados pelas equipes e os dois últimos os lances livres convertidos por elas respectivamente. Todas as entradas pertencem ao intervalo [1, 10]. O valor -1 na primeira posição de uma entrada indica o final dos casos de teste.

Saída

Para cada caso de teste deverão ser apresentadas quantas possíveis sequências de cestas existem para chegar ao placar final.

Exemplos de testes

Entrada	**Saída**
2 2 2 2 2 2 0 0 4 4 2 2 5 9 6 0 9 6 8 6 -1	6 2 180 0 0