MARATONA DE PROGRAMAÇÃO
InterFatecs

V MARATONA DE PROGRAMAÇÃO INTERFATECS 2016

http://interfatecs.fatecsp.br/

2ª Fase - 20 de agosto de 2016

Caderno de problemas – Prova principal COMENTADA

APOIO:

REALIZAÇÃO:

Centro Paula Souza

Fatec São Paulo

GOVERNO DO ESTADO SÃO PAULO
Secretaria de Desenvolvimento Econômico, Ciência, Tecnologia e Inovação

V MARATONA DE PROGRAMAÇÃO INTERFATECS 2016

Prova principal COMENTADA

Este caderno contém 11 problemas – identificados pelas letras de A até K, com páginas numeradas de 2 até 23. Verifique se o seu caderno está completo.

Informações gerais

A) Sobre o arquivo de solução e submissão:

1. O arquivo de solução (o programa fonte) deve ter o mesmo nome que o especificado no enunciado (logo após o título do problema).
2. Confirme se você escolheu a linguagem correta e está com o nome de arquivo correto antes de submeter a sua solução.

B) Sobre a entrada

1. A entrada de seu programa deve ser lida da entrada padrão (não use interface gráfica);
2. A entrada de um problema possui seus casos de teste segundo especificado no enunciado;
3. Em alguns problemas o final da entrada coincide com o final do arquivo. Em outros, o final é indicado por alguma quantidade de dados a serem lidos ou é especificado por alguma combinação de caracteres ou dígitos.

C) Sobre a saída

1. A saída do seu programa deve ser escrita na saída padrão;
2. Não exiba qualquer outra mensagem além do especificado no enunciado.

http://www.facebook.com/interfatecs/

Problem A

Antennas and more antennas!

Source file: `antenna.{c | cpp | java}`

Author: Silvio do Lago Pereira (Fatec São Paulo)

Tom is an excellent antenna installer and always does his best to make the customers happy. Tom usually attends orders in large buildings and he uses the stairs to move between the floors. For obvious reasons, he cannot use the elevators - they have little free space.

He has already visited buildings that do not have stairs between all floors. Because of that, he was not able to complete his work. Another problem he has already faced concerns obstacles that obstruct the passage in a floor, making it impossible to install an antenna. Then, before accepting the service, he analyses the building floor plan and define whether or not he is able to perform the service.

If the building has stairs, Tom can use them to move from a floor `i` to floor `i`+1 or to go down to floor `i`-1. At most, it may exist only one stair that connect a floor to the one right above it and one that connect a floor to the one right below it.

Tom will walk only vertically or horizontally, considering the perspective of a floor plan, never walking diagonally. He cannot cross obstacles – only free spaces or stairs can be reached. When in the stair he can also move up, down, left, or right in the same floor.

#	T	#	#	#	#
#	.	.	.	.	#
.	.	.	.	.	#
#	.	.	.	#	#
#	#	#	.	S	#
#	#	#	#	#	#

1st Floor

#	#	#	#	#	#
#	.	.	.	.	#
#	.	#	.	.	#
#	.	#	.	.	#
#	S	#	#	D	#
#	#	#	#	#	#

2nd Floor

#	#	#	#	#	#
#	.	.	.	A	#
#	#	#	#	.	#
#	#	.	.	.	#
#	D	.	#	#	#
#	#	#	#	#	#

3rd Floor

Figure 1: Floor plan of a regular building.

He needs your help to code a simple program that analyses a building floor plan and verify whether or not he can attend the service.

Input

The input has many test cases. The first row of each test case contains an integer N $(0 \leq N \leq 15)$, indicating the number of floors in the building, and an integer X $(0 \leq X \leq N)$, indicating the floor in which the antenna will be installed. The second row contains two integers A and L $(5 \leq A, L \leq 20)$ that indicate how many cells each floor has vertically and horizontally, respectively. The next $N \times A$ rows represents the floors with L characters. The character '#' represents an obstacle; '.' a free space; 'S' a stairs to upper floor; 'D' a stairs to lower floor; 'A' the spot where the antenna must be installed; 'T' the place where Tom starts – always in the 1st floor. The figure 1 illustrates the idea. The input ends when $N = 0$.

Output

For each test case, the program must print ‘viavel’ (lowercase, no quotes, nor special characters) when it is possible to install the antenna. Otherwise, the program must print ‘inviavel’ (lowercase, no quotes, nor special characters) and a line break;

Examples

Input:	**Output:**
3 3 6 6 #T#### #....# .....# #...## ###.S# ###### ###### #....# #.#..# #.#..# #S##D# ###### ###### #...A# ####.# ##...# #D.### ###### 4 3 5 7 ####### #.T...# ##.##.# #.S...# #.##.## ####### #####S# ##.##.# #.D..## ####### ####### #A...D# #.....# #S....# ####### ####### #..#..# #.....# #D...## ####### 0	viavel inviavel

Solução

Único problema em inglês do caderno da competição. É um problema de busca em labirinto, já cobrado em outras fases do InterFatecs. Os passos, superficialmente descritos, para resolver esse problema são os seguintes:

1) Alocar uma matriz de três dimensões, onde a primeira dimensão é o andar, a segunda dimensão é a altura do andar e a terceira dimensão é a largura do andar;

2) Enquanto o N dado de entrada não for zero, ler a altura e a largura;

3) Para cada andar, ler a planta dada como entrada;

4) Localizar a posição de Tom, fazendo uma simples varredura na matriz, no primeiro piso;

5) Criar uma função recursiva que faz buscas em uma matriz, percorrendo a partir do ponto em que Tom se localiza, inicialmente;

6) Se Tom não estiver no andar correto para instalar a antena, deverá procurar uma escada que dê acesso ao piso superior;

7) Assim que encontrar a escada a função recursiva deverá ser executada para o andar de cima, ou seja, a primeira dimensão da matriz deverá ser incrementada;

8) Já no andar correto, caso Tom consiga chegar até a posição marcada para a instalação, a função deverá retornar verdadeiro, indicando que foi viável executar o trabalho, caso contrário, retornará falso;

9) De posse do valor devolvido pela função, basta imprimir a mensagem indicada.

Problema B

Gráfico de Setores

Arquivo fonte: `setores.{c | cpp | java}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Manuela está trabalhando em um novo projeto para uma empresa de *coaching*. Trata-se de um sistema web responsivo que realiza um diagnóstico da inteligência emocional de uma pessoa qualquer.

Um dos requisitos funcionais especificados pelo cliente, o dono da empresa, é que o sistema produza um gráfico de setores (como um gráfico de pizza) com as competências do participante. O gráfico de setores envolve a divisão de uma região em setores proporcionais aos valores de suas competências. Como exemplo, assuma que o participante tem os seguintes valores de competências: 10, 20 e 30. Nele, existirão 3 setores, o primeiro deles ocupará 1/6 do tamanho da região (10 / (10+20+30)). O segundo ocupará 2/6 e o terceiro ocupará 3/6.

Para complicar, o requisito do cliente envolvia alguns detalhes. Os setores deveriam ser desenhados dentro de uma imagem: um corpo de um homem ou de uma mulher, dependendo do gênero da pessoa que esteja participando do diagnóstico. Ainda, os setores poderiam ser exibidos na horizontal (Figura 1 à esquerda) ou na vertical (Figura 1 à direita). Por fim, os setores deveriam ser exibidos na ordem do maior para o menor: da esquerda para a direita, na horizontal; de cima para baixo, na vertical.

Figura 1: Exemplos de gráficos de setores dentro de uma imagem: horizontal (à esquerda); vertical (à direita).

Como qualquer desenvolvedor faria, após identificar o requisito, Manuela acessou o Google e digitou diversas frases que pudessem levá-la a encontrar alguma biblioteca que a ajudasse com a tarefa. Infelizmente não encontrou. Desta forma, ela teve que desenvolver sua própria biblioteca.

Para deixar a solução genérica ela teve a ideia de permitir que o usuário da biblioteca utilize uma imagem qualquer. Sua ideia envolve visitar pixel por pixel da imagem e verificar se o pixel é transparente (fundo) ou não (imagem). Para desenhar o gráfico, ela considera apenas os pixels não transparentes. O tamanho de cada setor é definido pelo número de pixels que ele contém. Se, ao calcular a quantidade de pixels de um setor, o valor for um número de ponto flutuante, ela o arredonda para o menor inteiro mais próximo. Para o último setor (aquele com a menor quantidade de pixels), ela se certifica de que todos os pixels restantes da imagem sejam preenchidos. Isto é importante porque, como no cálculo do número de pixels do setor, ele considera o menor inteiro mais próximo, o último setor irá completar os pixels que restarem.

Uma mesma linha ou coluna de pixels da imagem pode conter cores de dois ou mais setores, conforme ilustrado no exemplo do enunciado. Para desenhar gráficos na horizontal, ela considera a ordem esquerda-direita e, em seguida, cima-baixo. Para gráficos na vertical, ela considera a ordem cima-baixo e, em seguida, esquerda-direita.

Após um dia longo de trabalho, ela batizou a biblioteca de SVGChart e a publicou no GitHub: https://github.com/leluque/svg-chart. Agora ela precisa realizar testes. Implemente uma solução que funciona de modo equivalente à biblioteca desenvolvida por ela e a ajude com os testes.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém dois números inteiros L ($5 \leq L \leq 100$) e C ($5 \leq C \leq 100$), separados por um espaço em branco, que representam o número de linhas e colunas da imagem, respectivamente. Seguem L linhas com C inteiros cada, representando o conteúdo da imagem. Um valor igual a 0 (zero) representa um pixel transparente de fundo. Um valor igual a 1 (um) representa um pixel de imagem. A próxima linha contém um caractere D, que representa a direção do gráfico: "V" (maiúsculo e sem aspas) para vertical e "H" (maiúsculo e sem aspas) para horizontal. A linha seguinte contém um número inteiro T ($1 \leq T \leq 9$), com o tamanho do conjunto de dados que será usado para calcular o tamanho dos setores. As próximos T linhas contém um inteiro W ($1 \leq W \leq 1000$) cada, representando um dado do conjunto de dados do gráfico. A entrada termina com o fim do arquivo.

Saída

Para cada caso de teste, deve-se imprimir uma linha contendo o texto "Caso I:" (sem aspas), onde I representa o número do caso de teste, iniciando em 1. Em seguida, devem ser impressas L linhas, com C inteiros cada, representando a imagem após o preenchimento dos setores. Cada setor deve ser representado por um número inteiro referente à sua posição no conjunto de dados de entrada (iniciando em 1). Após a última linha do último caso de teste, imprima uma quebra de linha.

Exemplos

Entrada	Saída
6 10 0000000000 0000110000 0001111000 0011111100 0111111110 0000000000 H 4 100 200 300 500 6 10 0000000000 0000110000 0001111000 0011111100 0111111110 0000000000 V 4 100 200 300 500	Caso 1: 0000000000 0000120000 0002233000 0033344400 0444444440 0000000000 Caso 2: 0000000000 0000340000 0002344000 0023344400 0123444440 0000000000

Solução

Trata-se de um problema de solução *ad hoc*. Para poder calcular a quantidade de pixels que cada setor ocupará, deve-se inicialmente contar a quantidade de pixels de imagem (com valor igual a 1). Isto pode ser feito durante a leitura da entrada. Em seguida, deve-se calcular a proporção de cada valor de setor em relação ao total. Para isso, soma-se os valores dos setores (o que pode também ser feito durante a leitura da entrada) e, para cada setor, divide-se o valor do setor pelo total. A partir das proporções, é possível calcular a quantidade de pixels que cada setor deve ter. Para tanto, ordena-se as proporções em ordem crescente e multiplica-se a proporção pela quantidade total de pixels. Para o último setor deste vetor, adiciona-se a quantidade de pixels restantes para completar a quantidade total de pixels de imagem. Por fim, caso os setores devam ser exibidos na horizontal.

Problema C

Azulejos

Arquivo fonte: `azulejos.{c | cpp | java}`

Autores: Anderson Viçoso de Araujo (UFMS)

A nova moda entre as madames da *high society* campo-grandense é decorar a suas cozinhas com azulejos coloridos. Marli, uma dessas distintas senhoras, quer a sua cozinha perfeita para poder mostrar para suas amigas do clube de campo. O problema é achar um pedreiro bom de serviço. Muitos deles, preenchem a sequência dos azulejos de forma incorreta, desrespeitando a ideia de cozinha perfeita para Marli.

Sua tarefa neste problema é escrever um programa que verifica se uma parede da cozinha é ou não perfeita para Marli e com isso ganhar uma graninha extra.

Uma parede com azulejos é representada por uma matriz de inteiros 9x9, onde cada inteiro de 1 a 9 representa um azulejo com um certo desenho. Para ser perfeita, cada linha e coluna deve conter todos os números de 1 a 9. Além disso, se dividirmos a matriz em 9 regiões 3x3, cada uma destas regiões também deve conter os números de 1 a 9. O exemplo abaixo mostra uma matriz representando uma parede que é uma solução do problema.

$$\left(\begin{array}{ccc|ccc|ccc} 1 & 3 & 2 & 5 & 7 & 9 & 4 & 6 & 8 \\ 4 & 9 & 8 & 2 & 6 & 1 & 3 & 7 & 5 \\ 7 & 5 & 6 & 3 & 8 & 4 & 2 & 1 & 9 \\ \hline 6 & 4 & 3 & 1 & 5 & 8 & 7 & 9 & 2 \\ 5 & 2 & 1 & 7 & 9 & 3 & 8 & 4 & 6 \\ 9 & 8 & 7 & 4 & 2 & 6 & 5 & 3 & 1 \\ \hline 2 & 1 & 4 & 9 & 3 & 5 & 6 & 8 & 7 \\ 3 & 6 & 5 & 8 & 1 & 7 & 9 & 2 & 4 \\ 8 & 7 & 9 & 6 & 4 & 2 & 1 & 5 & 3 \end{array}\right)$$

Entrada

São dadas várias instâncias. O primeiro dado é o número N ($0 < N < 100$) de matrizes na entrada. Nas linhas seguintes são dadas as N matrizes. Cada matriz é dada em 9 linhas, em que cada linha contém 9 números inteiros entre 1 e 9.

Saída

Para cada instância seu programa deverá imprimir uma linha dizendo "*Parede k*", onde *k* é o número do caso de teste atual. Na segunda linha, seu programa deverá imprimir "**SIM**" se a matriz correspondente a parede seja considerada perfeita para Marli, e "**NAO**" caso contrário. Imprima uma linha em branco após cada instância.

Exemplos

Entrada	Saída
2 1 3 2 5 7 9 4 6 8 4 9 8 2 6 1 3 7 5 7 5 6 3 8 4 2 1 9 6 4 3 1 5 8 7 9 2 5 2 1 7 9 3 8 4 6 9 8 7 4 2 6 5 3 1 2 1 4 9 3 5 6 8 7 3 6 5 8 1 7 9 2 4 8 7 9 6 4 2 1 5 3 1 3 2 5 7 9 4 6 8 4 9 8 2 6 1 3 7 5 7 5 6 3 8 4 2 1 9 6 4 3 1 5 8 7 9 2 5 2 1 7 9 3 8 4 6 9 8 7 4 2 6 5 3 1 2 1 4 9 3 5 6 8 7 3 6 5 8 1 7 9 2 4 8 7 9 6 4 2 1 3 5	Parede 1 SIM Parede 2 NAO

Solução

A solução para esse problema pode ser encontrada através da manipulação de matrizes e submatrizes. A ideia é verificar cada linha, coluna e a região de submatriz 3x3 e garantir que sempre correspondam ao critério, ou seja, que contenham sempre os números de 1 a 9. Se alguma dessas verificações for falsa a instância não corresponde a uma matriz de Marli.

Problema D

Salto em blocos

Arquivo fonte: `salto.{c | cpp | java}`

Autores: Lucas Tsutsui da Silva/Anderson Viçoso de Araujo (UFMS)

Gabriel é um garoto muito esperto e apaixonado pelos mais variados tipos de esportes. Em sua cidade, há uma competição todos os anos, em que os habitantes disputam diferentes modalidades de esportes típicos da região. A modalidade preferida de Gabriel é a chamada "salto em blocos". Este esporte é disputado em uma pista reta com um metro de largura, em que o comprimento da pista é dividido em blocos de um metro. Ou seja, a cada metro de pista, há uma linha que divide um bloco de outro, determinando assim o começo de um novo bloco.

Antes de começar a competição, os juízes sorteiam um número *N* para o competidor. Dado este número, o competidor deve percorrer toda a pista realizando saltos de tamanho *N*, ou seja, pulando *N*-1 blocos a cada salto, como exemplificado na Figura 1. Como os competidores são muito bem treinados, eles são capazes de pular uma grande quantidade de blocos a cada salto. Ao final da competição, ganha o competidor que conseguir finalizar a prova no menor tempo.

Figura 1: Exemplo de saltos de tamanho 2 em uma pista de 12 blocos.

Ao analisar bem a prova, Gabriel notou que após o término da competição, poderão existir blocos especiais por onde todos os competidores irão passar. Por exemplo, se apenas dois competidores disputaram a prova e um deles realizou saltos de tamanho 2 e o outro realizou saltos de tamanho 3, eles terão passado pelos blocos especiais nas posições 6, 12, 18, etc.

Gabriel ficou curioso para determinar quantos blocos especiais existirão ao final de uma competição e pediu sua ajuda para desenvolver um programa que o auxilie com essa tarefa.

Entrada

A entrada é composta por diversos casos de teste. Cada caso de teste se inicia por um inteiro positivo $\boldsymbol{x}$ ($x \leq 10^{15}$), que indica o tamanho da pista, e um inteiro positivo $\boldsymbol{y}$ ($y \leq 5$), que indica a quantidade de competidores. Após, seguem $\boldsymbol{y}$ linhas contendo um número positivo $\boldsymbol{n}$ ($n < 1000$), indicando o número sorteado para cada competidor, ou seja, qual será o tamanho dos saltos que ele dará. A entrada termina com fim de arquivo (EOF).

Saída

A saída para cada caso de teste deve conter um inteiro que determina a quantidade de blocos especiais existentes ao final da competição.

Exemplos

Entrada	Saída
12 2 1 2 500 5 2 3 4 5 7 100 3 2 5 11	6 1 0

Solução

Para resolver este problema, basta calcular o MMC (Mínimo Múltiplo Comum) entre todos os *y* números dados na entrada. Assim, temos um número que represente o período em que os blocos especiais irão se repetir. Para gerar a resposta basta dividir o tamanho da pista por esse número. Atentar- se para o tamanho de *x* : 10 15 só pode ser representado por um long long int em C e C++.

Problema E

Espetáculo

Arquivo fonte: `espetaculo.{c | cpp | java}`

Autores: Lucas Arakaki Takemoto/Anderson Viçoso de Araujo (UFMS)

Depois de ser reconhecida pelos melhores profissionais como a melhor cidade para pescar, Quixorá acabou ainda atraindo olhares de um dos grupos de espetáculos mais conhecidos do mundo: o *Unnamed Entertainment Group* (UEG). Isto aconteceu pois vários integrantes do grupo adoram pescar. Para tentar unir o útil ao agradável o grupo decidiu incluir a cidade de Quixorá em sua turnê pelo Brasil.

A UEG pode realizar quatro shows diferentes (3 normais e 1 especial), divididos por faixa etária da seguinte maneira:

- Apenas crianças e adolescentes de 6 a 17 anos (inclusive);
- Apenas adultos de 18 a 50 anos (inclusive);
- Pessoas **acima de** 50 anos (terceira idade);
- Sessão especial sem restrição de idade, caso necessário.

Sabendo que seus shows serão o maior sucesso, como sempre foram em qualquer lugar do mundo, os organizadores já esperam uma disputa muito grande pelos ingressos, que são limitados por dia da turnê. Assim sendo, os organizadores criaram as seguintes regras:

- Em um dia pode haver várias sessões;
- Para um determinado dia, apenas *N* ingressos serão liberados para venda;
- As *N* primeiras pessoas que chegaram à fila da bilheteria receberão um identificador inteiro, sequencial e crescente de 1 a *N*.
- Uma sessão normal só acontece se **exatamente** 5 pessoas forem designadas para ela. Para cada sessão normal específica, se menos de 5 pessoas forem designadas, essas pessoas participarão de uma **única sessão especial** sem restrição de idade;
- As sessões são intercaladas por faixa etária. A primeira sessão que acontece em um determinado dia é para crianças, a segunda para terceira idade, a terceira para adultos, a quarta para crianças novamente e assim por diante. Caso a sessão especial seja necessária, ela será a última do dia;
- Para cada sessão normal de uma determinada faixa etária, são designadas as 5 pessoas mais novas dessa faixa etária, por exemplo, no caso das crianças, para a primeira sessão, serão escolhidas as 5 crianças com menor idade. Caso não haja crianças suficientes para completar uma sessão, elas serão designadas para a sessão especial. Assim acontece com as outras sessões normais;
- Se uma pessoa já foi designada, ela sai da fila;
- Se existirem pessoas com a mesma idade na fila, deve ser seguida a ordem de inserção.

Deseja-se saber quantas sessões normais para cada faixa etária serão realizadas e se será necessário a realização de uma sessão especial. Se sim, quais são os identificadores dos "sortudos" que assistirão a sessão especial.

Entrada

A entrada contém vários casos de testes. Cada caso de teste representa um dia da turnê e consiste em duas linhas: a primeira contendo o número N ($N \leq 300$) de ingressos disponíveis; e a segunda contendo uma sequência F de N inteiros ($6 \leq F_1, F_2, \ldots, F_n \leq 85$) representado a idade de cada pessoa na fila. O último caso de teste é indicado quando $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste deverá ser impresso um identificador "#Dia X:" em que X é um número inteiro, sequencial e crescente a partir de 1. Logo após, o número de sessões que necessárias para crianças, terceira idade e adultos. Caso exista sessão especial, imprimir os identificadores, em ordem crescente e separados entre si por um espaço em branco, das pessoas que participarão da sessão. Caso contrário imprimir a mensagem "nao havera sessao especial".

Exemplos

Entrada	**Saída**
10 35 67 55 46 24 46 17 12 31 33 15 12 56 13 73 23 35 45 17 16 16 51 65 18 54 32 11 17 56 34 14 15 17 13 45 61 12 18 0	#Dia 1: 0 para criancas 0 para terceira idade 1 para adultos 2 3 6 7 8 #Dia 2: 1 para criancas 1 para terceira idade 1 para adultos nao havera sessao especial #Dia 3: 1 para criancas 0 para terceira idade 0 para adultos 2 3 6 8 9 11

Solução

A ideia principal para resolver este problema é ordenar a fila de acordo com a idade das pessoas e simultaneamente guardar o identificador de cada uma delas. Para isso foram utilizadas 3 listas, uma para crianças, uma para terceira idade e outra para adultos.

Ao ler cada pessoa da fila, armazenar na lista correspondente o valor da idade e do índice. Ordenar as 3 listas de acordo com a idade. Percorrer cada uma das listas verificando se cada uma tem mais pessoas que o tamanho limite (5), remover e adicionar em uma quarta lista que representa a sessão especial. Ordenar a quarta lista e imprimir os resultados.

Problema F

E agora Joãozinho?

Arquivo fonte: `padrao.{c | cpp | java}`

Autor: Julio Lieira (Fatec Lins)

Todo programador deve conhecer a charge onde a Professora, como castigo, pede para Joãozinho escrever 1000 vezes na lousa a frase "Proibido usar celular durante a aula" e Joãozinho, já um nerd aos 9 anos, se safou do castigo escrevendo na lousa o seguinte trecho de programa:

```
#include <stdio.h>
main(){
 int i;
 for (i=1;i<=1000;i++)
   printf("Proibido usar celular durante a aula.");
}
```

Mas o que ele não sabia é que sua Professora era campeã de Maratonas de Programação. Então, a Professora lançou o seguinte desafio: vou lhe apresentar um texto composto por um padrão de caracteres que se repete *X* vezes e você deve fazer um programa que detecte este padrão e escreva-o na tela, juntamente com a quantidade de vezes que ele apareceu. Note que este padrão deve ser o menor que repetido *X* vezes completa o texto. E agora Joãozinho!!!

Entrada

Cada linha da entrada é um caso de teste e representa o texto que contém o padrão que se repete. Este texto pode apresentar de 2 a 1000 caracteres, que podem ser letras do alfabeto, dígitos ou espaço. Considere caracteres maiúsculos diferentes de minúsculos.

Saída

Para cada caso de teste, deve-se imprimir uma linha contendo o número de vezes que o padrão se repete no texto, seguido de um espaço, seguido do caractere * (asterisco), seguido de outro espaço e finalizando com o padrão de caracteres que se repete. Após a última linha do último caso de teste, imprima uma quebra de linha.

Exemplos

Entrada	**Saída**

`EspertinhoEspertinhoEspertinho`
`abcdabdcxyaklabcdabdcxyaklabcdabdcxyaklabcdabdcxyakl`
`abababababab`
`aaaaaa`
`abababcdcdefefabababcdcdefef`
`abcdefghijklmnop`
	`3 * Espertinho`
	`4 * abcdabdcxyakl`
	`5 * ab`
	`6 * a`
	`2 * abababcdcdefef`
	`1 * abcdefghijklmnop`

Solução

Embora este problema seja facilmente resolvido com expressões regulares em Java, em C será preciso montar o padrão caractere por caractere. A seguir os passos de uma possível solução iterativa:

1) Leia o texto em uma string (Ex.: `strtexto[]=EspertinhoEspertinhoEspertinho`)

2) Declare uma outra string para armazenar o padrão, do mesmo tamanho do texto (`strpadrao[]`)

3) Pegue o primeiro caractere de `srttexto[]` e coloque na string padrão (Ex.: `strpadrao[]=E`)

4) Vá atribuindo os caracteres do texto para a string padrão, até encontrar um caractere igual ao primeiro caractere do texto (ex.: `strpadrao[]=Espertinho`)

5) Verifique se esta string `strpadrao[]` se repete no texto a partir desta posição até o final, contando quantas vezes ela se repete. Se conseguir checar até o final, imprima a quantidade de vezes e a string padrão. Senão, continue atribuindo novos caracteres do texto ao padrão até encontrar novamente um caractere igual ao primeiro caractere. Repita o processo.

Problema G

Fox

Arquivo fonte: `fox.{c | cpp | java}`

Autor: Lucio Nunes de Lira (Fatec São Paulo)

Megan é uma aluna de computação que também é atriz em grandes filmes de ação. Buscando se livrar do estigma de trabalhar apenas em obras cinematográficas com robôs gigantes que se destroem, resolveu fazer a inscrição para um papel em que interpretará uma brilhante professora de matemática, que também é o papel principal.

O que Megan não sabia é que o diretor, responsável pela palavra final na contratação do elenco, propôs a todas as candidatas à vaga de protagonista um problema envolvendo matemática, que decidirá quem irá para a próxima fase da audição.

No caso de nossa amiga, o problema oferecido é o seguinte:

- O diretor irá passar uma sequência de números naturais em base decimal;
- Megan deverá guardá-los de alguma forma;
- Para cada valor informado Megan deverá dizer:
    a) A palavra “primo”, caso o valor seja um número primo, ou;
    b) O número primo imediatamente anterior e o número primo imediatamente posterior a esse valor, caso ele não seja um número primo.

É sempre bom lembrar que um número primo é um número natural que tem exatamente dois divisores: o número um e ele próprio. O número um não é primo, já o número dois é o único primo par.

Sabendo que Megan é aluna de um curso onde há a disciplina de Algoritmos e Lógica de Programação, o gentil diretor permitiu o uso de um computador para auxiliá-la durante a resolução do problema.

Você, que é colega de sala de Megan e tem grandes habilidades em programação, se propôs a construir um programa que, dados os valores passados pelo diretor, processa e mostra o resultado pedido. Em troca de sua grande ajuda, Megan lhe dará um presente muito especial! Dois ingressos em lugares privilegiados para assistir a estreia do filme, caso ela ganhe o papel.

Entrada

A entrada contém vários casos de teste. Cada linha de cada caso de teste contém um inteiro N ($1 \leq N \leq 100.000$), indicando a quantidade de valores que serão informados pelo diretor. Seguem-se então N linhas, cada uma contendo um valor inteiro V ($2 \leq V \leq 1.000.000$) que é o número natural em base decimal passado pelo diretor. A entrada termina com o fim do arquivo.

Saída

Para cada caso de teste, deverá ser impressa a frase `"CASO #:"`, onde `"#"` é o número do caso de teste; nas linhas abaixo, para cada valor `V` da entrada, imprima `"X e Y"` caso o `V` seja primo, onde `X` é o número primo imediatamente anterior e `Y` o número primo imediatamente posterior a `V`, ou `"primo"`, caso o `V` seja um número primo, e uma quebra de linha.

Exemplos

Entrada	Saída
3 18 3 13 2 6 999611 1 2	CASO 1: 17 e 19 primo primo CASO 2: 5 e 7 primo CASO 3: primo

Solução

Parece que muitos conseguiram ajudar a senhorita Megan! O que deixou os competidores e a fictícia atriz bastante contentes, exceto aqueles que não perceberam que o problema se tratava essencialmente de desempenho e atenção nos detalhes.

Os principais pontos para a resolução desse problema são os que seguem:

1) Alocar um vetor de números inteiros que comportasse toda a faixa de valores que poderiam ser perguntados a Megan. Esse ponto é essencialmente curioso, uma vez que muitos poderiam pensar que alocar um milhão de posições seria o bastante para isso, quando na verdade deveria ser percebido que esse valor é o maior que pode ser perguntado a ela. Logo, como não é primo, o próximo primo após um milhão deveria ser informado também;

2) Guardar todos os valores primos, calculando-os apenas uma vez. Provavelmente, dado o limite de tempo, processar para cada entrada os valores necessários para verificar se um valor é ou não primo consumiria um tempo inviável na competição;

3) Ter um algoritmo razoavelmente rápido para o calculo de primos dentro de uma faixa de valores bem definida. Existem várias formas de saber se um valor é ou não primo, a questão é se o procedimento escolhido é eficiente. O Crivo de Eratóstenes seria bom o suficiente.

Problema H

Explosão de cores

Arquivo fonte: `explosao.{c | cpp | java}`

Autor: Marcelo Duduchi (Fatec São Paulo)

"Sou um artista incompreendido!". Esse é o pensamento de Raphael, um pintor que tem características, digamos, peculiares para montar suas obras de arte. Seu foco é inovar! Sempre que possível inventa uma técnica artística e quanto mais exótica melhor… para ele.

O procedimento mais recente elaborado por Raphael é um estouro! Sério, literalmente:

- Apoia-se uma tela de pintura transparente quadriculada em uma mesa;
- Põe-se uma bombinha de tinta, projetada por Raphael, em um dos quadrados;
- A bombinha explode e colore uma porção da tela com uma cor específica;
- Repete-se o procedimento com outras bombinhas, em quadrados diferentes ou não, até que Raphael esteja satisfeito e grite "Eureka!".

Mesmo sendo um artista, Raphael também é aluno do curso de Análise e Desenvolvimento de Sistemas (e isso não o torna mais artista ainda?!) e gosta de saber detalhes matemáticos das obras que cria. Nesse caso, dada uma sequência de bombinhas que serão usadas, está interessado em saber, em ordem decrescente, a predominância de cores final de sua tela. Porém, existem detalhes:

- Toda bombinha possui duas características: a cor da tinta e a potência da explosão, que indica quantos quadrados da tela serão preenchidos;
- Poderão existir diversas bombinhas com a mesma cor e até vinte cores diferentes;
- As bombinhas espalham tinta somente em cruz, ou seja, na vertical e na horizontal, inclusive considerando o ponto onde foi colocada, jamais pintando as diagonais;
- A tinta de uma nova explosão sobrepõe a de uma antiga, caso as áreas coincidam.

Raphael é um artista muito ocupado, por isso você, sábio aluno da Fatec, decidiu ajudá-lo. Sua missão é construir um programa que, dados os nomes das cores das tintas das bombinhas, as potências de explosão e as posições dos quadrados onde serão colocadas as bombinhas, exibe, em ordem decrescente de predominância, os nomes das cores (exatamente como escritas) e a quantidade de quadrados que foram coloridos por cada cor. Ideia ilustrada, simplificadamente, na Figura 1.

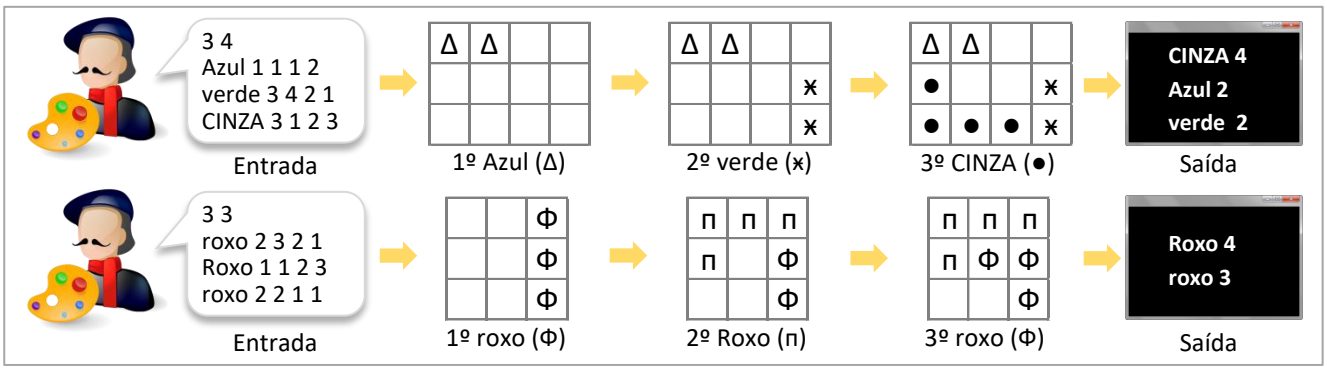

Figura 2: Exemplo de interação de Raphael com o sistema.

Entrada

A entrada contém vários casos de teste. Cada linha de cada caso de teste contém dois inteiros TA e TL ($1 \leq TA, TL \leq 100$), indicando a altura e a largura da tela de pintura que será usada; na linha seguinte, o nome de uma cor C ($1 \leq caracteres(C) \leq 20$), podendo conter *underlines* e letras não acentuadas, maiúsculas e minúsculas; dois inteiros QV ($1 \leq QV \leq TA$) e QH ($1 \leq QH \leq TL$), que indicam a posição vertical e horizontal do quadrado em que a bombinha será colocada, respectivamente; além de dois inteiros, PV e PH ($1 \leq PV, PH \leq 50$), indicando a potência vertical e horizontal de cada bombinha, respectivamente. A entrada de cada caso de teste termina quando a cor C lida for "EUREKA" (sem aspas e maiúsculo) e a entrada total termina com o fim do arquivo.

Saída

Para cada caso de teste deverá ser impresso, em ordem decrescente de predominância (em empates prevalece a ordem de entrada), o nome da cor, a quantidade de quadrados coloridos com a respectiva cor e uma quebra de linha, para todas as cores usadas por Raphael. Imprima uma linha em branco após cada caso.

Exemplos

Entrada	**Saída**
3 4 Azul 1 1 1 2 verde 3 4 2 1 CINZA 3 1 2 3 EUREKA 3 3 roxo 2 3 2 1 Roxo 1 1 2 3 roxo 2 2 1 1 EUREKA	CINZA 4 Azul 2 verde 2 Roxo 4 roxo 3

Solução

Apesar de detalhado e com enunciado extenso, é um problema relativamente fácil de ser implementado, não envolvendo técnicas avançadas de programação.

Basicamente o que precisava ser feito era:

1) Alocar uma matriz com tamanho suficiente para representar a tela de Raphael;

2) Ler cada uma das cores até que a palavra EUREKA fosse apresentada, o que poderia ser facilmente verificado com uma função pronta da linguagem escolhida (e.g. strcmp() em linguagem C);

3) Armazenar a cor da bombinha em um vetor de strings, com especial atenção para não duplicar cores exatamente iguais;

4) Colorir a matriz de acordo com as potências das bombinhas, com ressalvas para não ultrapassar os limites da tela, usando quatro laços de repetição, cada um representando uma das possíveis direções;

5) Contabilizar o quanto de cada cor ficou na tela, inclusive as cores que foram completamente sobrepostas por outras;

6) Realizar uma ordenação simples, na ordem solicitada no enunciado;

7) Exibir as cores e a quantidade de quadrados que cada uma cobriu da tela.

8) Um dos maiores cuidados era em relação ao término da entrada, que é descrito claramente como EOF (fim da entrada principal) e como EUREKA (fim do caso de teste).

Problema I

Hora de aposentar

Arquivo fonte: `aposenta.{c | cpp | java}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

As regras de aposentadoria no Brasil são uma eterna fonte de dúvidas, confusão e polêmica. É um emaranhado de situações especiais, direitos adquiridos e detalhes técnicos que tornam difícil para um cidadão determinar exatamente quando ele poderá se aposentar. No momento está em vigor a chamada regra 85 / 95, segundo a qual uma mulher pode se aposentar sem qualquer prejuízo de seu benefício quando a soma entre sua idade com o tempo de contribuição à Previdência Social atingir 85 anos. Para os homens essa soma precisa atingir 95 anos. Uma revisão dessa regra propõe que esse total de anos a ser atingido na soma da idade com o tempo de contribuição se modifique progressivamente com o tempo, até que a regra se torna 90 / 100, conforme mostra a tabela a seguir.

Ano	**Idade + Contribuição (Homem)**	**Idade + Contribuição (Mulher)**
Até 2018	95	85
2019	96	86
2020	96	86
2021	97	87
2022	97	87
2023	98	88
2024	98	88
2025	99	89
2026	99	89
2027 em diante	100	90

Para entender a tabela, considere um trabalhador (gênero masculino, portanto) que em 2016 tem 51 anos de idade e 35 anos de tempo de contribuição à Previdência Social. Ele só poderá se aposentar pela regra em questão no ano de 2022, pois nesse momento terá 57 anos de idade e 41 anos de tempo de contribuição, totalizando 98 anos, que igual ou superior ao limite mínimo desse ano, que para homens é 97 anos. Uma trabalhadora que em 2016 tem 49 anos de idade e 33 anos de tempo de contribuição deverá se aposentar em 2018, pois atingirá nesse ano um total combinado de 86 anos (51 de idade + 35 de contribuição). Obviamente estamos considerando sempre que o trabalhador continuará ativo ininterruptamente até o momento em que se aposentar.

O seu objetivo neste problema é produzir um aplicativo que, apenas com base no tempo de contribuição de uma pessoa ainda não aposentada (em anos), a sua idade (também em anos) e a indicação se trata-se de um homem ou de uma mulher, determina em que ano ela poderá se aposentar segundo as regras descritas anteriormente.

Entrada

O conjunto de dados a serem processados pelo aplicativo é iniciado com um inteiro N ($1 \leq N \leq 300$), indicando a quantidade de casos de teste. Seguem-se então N linhas, cada uma composta por um inteiro I ($1 \leq I \leq 100$), representando a idade (em anos completos) do indivíduo em 2016, um inteiro C ($0 \leq C \leq 50$), que é o tempo de contribuição à Previdência Social totalizado em anos completos até 2016 e um caractere G indicando o gênero do indivíduo, onde 'M' indica gênero masculino e 'F' corresponde a gênero feminino. Esses três valores são separados entre si por um espaço em branco.

Saída

Imprima o ano a partir do qual a pessoa poderá se aposentar, com quatro dígitos, seguido de uma quebra de linha.

Exemplos

Entrada	**Saída**
2 51 35 M 49 33 F	2022 2018

Solução

É um problema de simulação. Crie e alimente inicialmente uma matriz que implemente a informação disponível na tabela fornecida no enunciado. Você pode fazer isso em uma tabela só ou em duas tabelas, uma para cada sexo.

Para cada indivíduo informado, some sua idade e tempo de contribuição e assuma o ano 2016. Em seguida, inicie a simulação, verificando na matriz se já é possível a aposentadoria com os limites de 2016. Caso seja, imprima o ano e prossiga com o próximo indivíduo. Caso não seja, some um no ano, um na idade e um no tempo de contribuição e volte a testar.

Problema J

Linguagem

Arquivo fonte: `linguagem.{c | cpp | java}`

Autor: Danilo Ruy Gomes (Fatec Itapetininga)

Atualmente existem diversas linguagens de programação. Discutir qual é a melhor é a mesma coisa que discutir time de futebol, religião ou política, cada um tem suas preferências por conveniência e ponto final.

Robinho um mestrando em Ciência da Computação resolveu colocar mais lenha nessa fogueira e resolveu criar uma nova linguagem de programação. A princípio, sua linguagem será utilizada somente para imprimir caracteres na tela, que envolverá comandos de repetição e impressão.

Basicamente, esta nova linguagem funcionará com alguns comandos como demonstrado na tabela abaixo:

Comando	**Funcionalidade**
`<I>LETRA`	Insere na cadeia o caractere indicado na sequência
`<A>`	Apaga o último caractere inserido na cadeia
`<L>`	Repete o último caractere inserido na cadeia
`<IS>`	Imprime toda a cadeia, a partir de seu início
`<II>`	Imprime toda a cadeia, a partir de seu final
`<IN>P`	Imprime a cadeia sequencialmente, a partir de seu *P*-ésimo caracter
`<IL>N`	Imprime a cadeia, a partir de seu início, *N* vezes
`<F>`	Fim do programa

Todo comando deve ser especificado com o caracter ‘<’ em seu início e com o caracter ‘>’ em seu final. Caso o comando requeira algum parâmetro numérico, este virá após o ‘>’ final. Um exemplo de entrada de comandos seria como o demonstrado abaixo:

Comandos	**Impressão**
`<I>A<I>B<IS><F>`	`AB`
`<I>A<I>B<II><F>`	`BA`

Os comandos de impressão e repetição poderão ser inseridos em qualquer ordem. O comando `<F>` deverá ser inserido quando o programa for encerrado. Leve em consideração que os comandos deverão ser inseridos com letras maiúsculas e iniciados a partir da posição 0 (zero). Qualquer espaço em branco deverá ser ignorado.

Com base nas tabelas acima anteriores, você aluno de ADS que já cursou Estrutura de Dados, conseguiria ajudar Robinho a resolver o problema?

Entrada

A entrada consiste em vários casos de teste, cada um deles sendo dado por uma linha contendo uma sequência de C caracteres ($10 \leq C \leq 500$) onde são informadas as instruções de sua linguagem que devem obrigatoriamente terminar com instrução `<F>`. O conjunto de entradas é encerrado com o fim do arquivo (EOF).

Saída

Seu programa deve imprimir uma única saída para cada caso de teste após informada a instrução de impressão.

Exemplos

Entrada

```
<I>F<I>A<I>T<I>E<I>C<I>I<I>T<I>A<IS><F>
<I>F<I>A<I>T<I>E<I>C<I>I<I>T<I>A<II><IL>3<F>
<I>D<I>E<I>R<L><L><IS><II><IS><IS><IN>3<F>
```

Saída

```
FATECITA
ATICETAFFATECITAFATECITAFATECITA
DERRRRRREDDERRRDERRRRRR
```

Solução

Problema envolvendo manipulação de cadeias. O objetivo era receber uma sequência de caracteres e, segundo o especificado em seu conteúdo, realizar a instrução de acordo com a tabela de comandos. Uma string inicialmente vazia deveria ser alterada conforme as instruções de atribuição e exclusão fornecidas na entrada, as saídas correspondem ao que foi impresso pelas instruções específicas de impressão.

Este problema também poderia ser resolvido através de estruturas envolvendo listas encadeadas.

Problema K

OpenGL txt ?

Arquivo fonte: `linha.{c | cpp | java}`

Autor: Julio Lieira (Fatec Lins)

Na Fatec Lins tem um professor que os alunos dizem ser "maluco". Ao invés de utilizar os recursos do motor de Física do Unity, ele insiste em implementar um motor de Física usando OpenGL. Diz ele "Assim vocês aprendem os fundamentos de um motor de física e depois saberão utilizar melhor os recursos da ferramenta". Se não bastasse, agora ele colocou na cabeça que "poderíamos implementar um OpenGL para rodar no prompt de texto do Windows, tipo um OpenGLtxt". E decidiu iniciar implementando a versão txt da função glLineStipple(fator,padrão), a qual determina o estilo da linha que será traçada pelas funções de desenho de linhas (ou retas). Como exemplo, considere a função com os seguintes argumentos:

```
glLineStipple(1,0xF6DE);
```

O padrão 0xF6DE está em hexadecimal e deve ser convertido em binário, obtendo-se o valor 1111011011011110 (note que o 0x indica um valor hexadecimal e deve ser desprezado na conversão). Este valor especifica a máscara que determina quais bits estarão ligados (1) e quais estarão desligados (0), e deve ser interpretado de trás para frente. A Figura 1 ilustra a aplicação deste padrão no estilo da linha. Note que cada bit 1 representa um pixel ativado e cada bit 0 representa um pixel desativado.

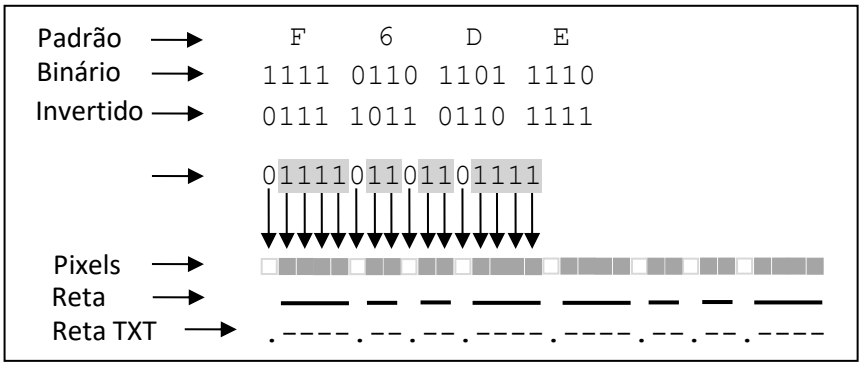

Figura 1: Procedimento OpenGL para Estilo de Linhas.

Neste exemplo foi aplicado um fator de multiplicação de 1. Aplicando-se um fator 2, por exemplo, faz com que a quantidade de bits 1 e 0 no padrão seja multiplicada por 2. Assim, teríamos 2 pixels desativados, seguidos de 8 pixels ativados, seguido de 2 pixels desativados, depois 4 pixels ativados, depois 2 pixels desativados, depois 4 pixels ativados, seguidos de 2 pixels desativados e finalizando com 8 pixels ativados. Ou seja, o fator aumenta o tamanho dos traços e espaços entre os traços.

Sua tarefa é implementar a versão txt da função. Assim, dado o fator e o padrão, seu programa deverá imprimir a reta seguindo os procedimentos descritos. Porém, como a interface de saída é texto e não gráfica, ao invés de utilizar pixels, você deverá imprimir um hífen (-) para representar um pixel (bit) ativado e um caractere ponto (.) para representar um pixel (bit) desativado.

Entrada

Cada linha da entrada é um caso de teste contendo um valor inteiro F ($1 \leq F \leq 10$), seguido de um espaço, seguido de 4 dígitos hexadecimais (0, 1, 2, 3, ..., 9, A, B, C, D, E, F) os quais representam o padrão da linha. Considere que os caracteres dos dígitos em hexa sempre aparecem em maiúsculo.

Saída

Para cada caso de teste deve-se imprimir uma reta (linha) de tamanho 60 caracteres no estilo proposto pelo fator e o padrão da entrada. Lembre-se de usar hífen (-) para um pixel ativado e ponto (.) para um pixel desativado. Após a última linha do último caso de teste, imprima uma quebra de linha.

Exemplos

Entrada	Saída
1 F6DE	.----.--.--.----.----.--.--.----.----.--.--.----.----.--.--.
2 F6DE	..--------..----..----..--------..--------..----..----..----
1 F020	.....-......----.....-......----.....-......----.....-......
1 5555	-.
3 5555	---...---...---...---...---...---...---...---...---...---...
1 ABAF	----.-.---.-.-.-----.-.---.-.-.-----.-.---.-.-.-----.-.---.-

Solução

Principais Passos para a solução do Problema:

1) Converter a string hexadecimal em binário. Para tanto, uma maneira simples é converter cada dígito hexadecimal em binário usando 4 bits. Com isso teríamos:

Hexa	Binário
0	0000
1	0001
2	0010
3	0011
4	0100
...	...
9	1001
A	1010
B	1011
...	...
F	1111

Assim, crie uma string com os dígitos binários.

2) Inverta a string de dígitos Binários

3) Supondo que `strbinario[]` é a string de dígitos binários já invertida e fator é o fator lido na entrada, o trecho de código abaixo faz a impressão da linha com tamanho 60:

```
//Imprime Linha
k=0;
for (i=1;i<=60;i++){
    if (strbinario[k]=='1')
        ch='-';
    else
        ch='.';
```

```
    f=1;
    while ((i<=60)&&(f<=fator)){
        printf("%c",ch);
        f++;
        i++;
    }
    i--; //Fez 1 no while e vai fazer outro no for
    k=(k+1)%16;
}
printf("\n");
```