MARATONA DE PROGRAMAÇÃO

InterFatecs

V MARATONA DE PROGRAMAÇÃO INTERFATECS 2016

http://interfatecs.fatecsp.br/

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

1ª Fase - 21 de maio de 2016

Caderno de problemas

Este caderno contém 10 problemas – identificados por letras de A até J, com páginas numeradas de 2 até 17. 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 é composta por vários casos de teste, conforme especificado no enunciado de cada problema;
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.

Promoção:

FUNDAÇÃO FAT

Fatec São Paulo

CPS Centro Paula Souza

ENIAC Educação Básica e Superior

Camisetas evidence

SBC Sociedade Brasileira de Computação

mf Soluções em Tecnologia

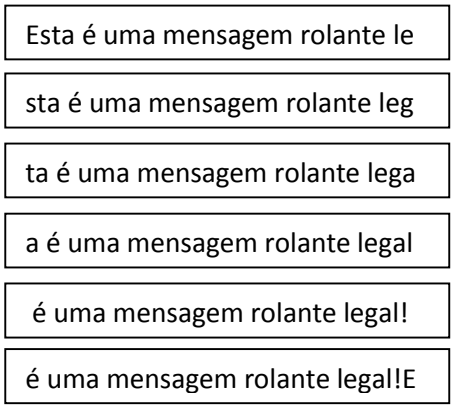
Problema A

Marqueenho

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

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Marquinho é um desenvolvedor Web das antigas - começou na época em que Internet Explorer (IE) e Netscape Navigator (NN) protagonizavam a batalha dos navegadores. Embora o pai dele insistisse para que ele usasse o NN, ele gostava mesmo era do IE. Ele dizia: - "Pai, a versão atual do NN ainda não implementa a marcação (*tag*) HTML marquee". A *tag* HTML marquee foi um recurso introduzido nas primeiras versões do IE que criava um texto rolante na tela, conforme ilustrado na parte esquerda da Figura 1.



OBS: Um marquee tem um tamanho, que especifica o número de caracteres que cabem nele. No exemplo à esquerda, o tamanho é 30. O texto do marquee é exibido alinhado à esquerda. Caso o texto não caiba por inteiro no marquee, a parte direita dele é ocultada. A cada instante de tempo, o conteúdo do marquee é deslocado um caractere para a esquerda. Quando uma palavra inteira for deslocada para à esquerda (uma palavra inteira é considerada qualquer sentença separada de outras por um espaço em branco ou uma pontuação ",.:;!?"), ela é inserida no final do texto do marquee.

O Marquinho adorava a marquee. Em todo site que criava, utilizava uma, fosse para exibir notícias ou uma simples mensagem de boas-vindas rolante. Ele era tão viciado na *tag*, que os colegas começaram a chamá-lo de Marqueenho. Ele não ligava.

Até que o W3C, entidade responsável pela padronização da Web, classificou a *tag* como não recomendada e os amigos de Marquinho aproveitaram para fazer uma brincadeira com ele. Eles disseram que o IE deixaria de implementar o marquee.

Poxa! O menino perdeu as estribeiras, saiu do eixo. Aqueles foram dias difíceis. Com o apoio de um terapeuta, ele passou por todas as fases comuns ao luto. Primeiro, a negação. Para os amigos, ele dizia que era temporário, que a *tag* voltaria na próxima versão do IE. Depois, a raiva. Enviou cartas xingando o Bill Gates e sua família e, dizem, chegou até a criar um site associando o número 666 ao nome completo de Bill Gates, quando considerada a codificação da tabela ASCII. Em seguida, veio a negociação, a fase que nos interessa. Ele colocou na cabeça que o marquee não seria retirado do IE se ele fizesse uma boa implementação para este recurso. Na cabeça dele, aquilo estava acontecendo porque deveria haver alguma coisa errada com o código que exibia o marquee no IE. Por isso, ele pediu a sua ajuda para testar se o algoritmo que ele está implementando está correto.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém dois números inteiros $M(10 \leq M \leq 100)$ e $T(10 \leq T \leq 300)$, separados por um espaço em branco, que representam o tamanho do marquee e o tamanho do texto (conteúdo do marquee) em caracteres, respectivamente. A próxima linha contém um texto de tamanho T. A última linha de cada caso de teste contém um número inteiro $E(0 \leq E \leq 1000)$, que representa a quantidade de unidades de tempo transcorridas após a exibição do marquee.

Saída

Para cada caso de teste, deve-se imprimir o conteúdo atualmente em exibição no marquee após a passagem das *E* unidades de tempo especificadas na entrada – a cada unidade de tempo, o marquee move um caractere para à esquerda. Caso o marquee seja maior que o texto, o restante dele à direita ficará vazio. Quando uma palavra completa sumir à esquerda, ela será colocada imediatamente após o último caractere do texto conteúdo do marquee.

Exemplos

Entrada	**Saída**
5 9 A bc C 1. 1 5 9 A bc C 1. 3 5 9 A bc C 1. 4 5 9 A bc C 1. 5	bc C c C 1 C 1. C 1.A

Solução

Trata-se de um problema simples de manipulação de strings. Uma solução possível seria simular a movimentação do marquee, por meio de uma repetição, sem se preocupar com o tamanho dele. Ao término da simulação, bastaria exibir os primeiros *M* caracteres do conteúdo atual do marquee. Outra possível solução seria calcular (por meio do resto de divisão) o caractere inicial do conteúdo do marquee no instante *E*. A partir deste caractere, bastaria navegar para a direita do conteúdo (e, se necessário, retornar ao início do conteúdo) até que uma string de tamanho *M* fosse criada.

Problema B

Escolha científica

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Geraldo é um dos vendedores da empresa XPTO S.A. e seu trabalho envolve visitar clientes com uma certa frequência. Como foi reconhecido como o "Vendedor do Ano" da empresa no ano passado, ganhou o direito de escolher sempre o veículo da frota da empresa que utilizará em seus deslocamentos. Os veículos ficam no estacionamento da XPTO mas estão à disposição daqueles que necessitarem fazer alguma viagem a trabalho. O funcionário escolhe um dos veículos disponíveis, pega a chave com o garagista, assina o termo de responsabilidade e sai para cumprir suas obrigações. Cada veículo possui um número de 1 até *N*, onde *N* é a quantidade de veículos da frota. Na volta o funcionário estaciona o veículo, tranca suas portas, entrega a chave ao garagista e assina o registro de controle correspondente.

Apesar de ser um bom vendedor, Geraldo é uma pessoa com dificuldades em escolher coisas triviais no dia a dia. Tem problema com isso desde pequeno e, para escapar de situações que envolvem escolha entre múltiplas alternativas, ele usa um truque de criança: vai recitando baixinho "Mi-nha-mãe-man-dou-eu-es-co-lher-es-te-da-qui-mas-co-mo-sou-de-so-be-di-en-te-vou-es-co-lher-es-te-da-qui", enquanto a cada pedaço desse mantra ele vai apontando para uma das alternativas. Se antes do fim do mantra ele chega ao último elemento da lista de alternativas, ele volta a apontar para o primeiro elemento e continua o processo normalmente. Quando a última parte do mantra é recitada (a sílaba "qui" final), a alternativa para a qual ele estiver apontando será a escolhida. Como a empresa possui uma quantidade relativamente grande de carros, ele quer um programa que, dada a quantidade de veículos disponíveis, informe qual ele deverá escolher com base em seu critério altamente científico.

Entrada

A entrada consiste de vários casos de teste. Cada caso é dado por um inteiro $N (2 \leq N \leq 65)$ que indica a quantidade de veículos disponíveis para Geraldo escolher. Quando um valor $N = -1$ for lido da entrada, o programa deverá encerrar o processamento.

Saída

Para cada caso de teste imprima o inteiro correspondente ao veículo que Geraldo deveria escolher, seguido de uma quebra de linha. Considere que todos os veículos de 1 a N estarão disponíveis.

Exemplos

Entrada	**Saída**
8 12 16 -1	7 7 15

Solução

Este era, talvez, o problema mais fácil da prova. O mantra repetido por Geraldo possui 31 partes. Se a quantidade de veículos disponíveis, indicada em *N*, for maior que 31, será escolhido o veículo 31; se temos menos de 31 veículos, após chegar ao último, continuamos o mantra novamente a partir do primeiro. Assim, se temos 28 veículos, a 29ª sílaba do mantra corresponde ao primeiro veículo, a 30ª corresponde ao segundo e a 31ª ao terceiro, ou seja, o veículo escolhido é dado pelo resto da divisão de 31 por *N*. Assim, bastava receber

o valor de *N* e fazer a divisão de 31 por esse valor: se o resto for zero, o veículo escolhido é o 31, senão, é o veículo correspondente ao próprio resto.

Problema C

Vacina

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

Autor: Danilo Ruy Gomes (Fatec Itapetininga)

Os laboratórios do mundo inteiro andam em polvorosa a busca de uma vacina que consiga combater as três doenças transmitidas pelo mosquito *Aedes aegypt*: *chikungunya*, dengue e *zika* vírus. Uma das pesquisas concentra-se em encontrar uma subsequência de DNA contida nos três vírus, que possam ser estudados e consequentemente achar um padrão para a criação de uma vacina

Francisco Bento, é formado em biomedicina e trabalha em um desses laboratórios, porém em conversa com seu filho Bernardo que estuda TI na FATEC ele descobriu que tais padrões são uma sequência de letras que podem ou não se repetir. Por exemplo a subsequência "**AAGG**" é um segmento da palavra AA**AAGG**CC, enquanto a subsequência **TTTG** não é encontrado na sequência GGAAGGCC. Assim para mapear o DNA dos três vírus, tais subsequências devem ser encontradas em três palavras diferentes, de acordo com um comprimento T. Por exemplo, se Francisco decidir que $T = 3$, então ele considera "**AGG**" como uma subsequência comum aceitável de A**AGG**BDDD, **AGG**GGGGG e GAT**AGG**CC. Neste caso **AGG** foi encontrando somente **uma vez** nos três filamentos de DNA. Se Francisco decidir que o tamanho da subsequência $T = 2$ onde para dengue com filamento de DNA **AAG**G, chikungunya **AAG**T e zica **AAG**C temos que a subsequência **AA** se repete nos 3 filamentos, assim como **AG**, neste caso teremos duas subsequências se repetindo nos 3 filamentos.

Com base nesta ideia, você poderia ajudar Bernardo a criar um algoritmo que encontre a quantidade de subsequências que se repetem nos três vírus?

Entrada

A entrada consiste de vários casos de testes. A primeira linha de cada caso de teste contém uma cadeia de caracteres indicando um filamento de DNA do chikungunya com o tamanho $X (4 \leq X \leq 15)$. A segunda linha contém outro filamento de DNA da dengue com tamanho $D (4 \leq D \leq 15)$ e a terceira linha contém outro filamento de DNA para o zica com tamanho $Z (4 \leq Z \leq 15)$. A última linha contém o tamanho da subsequência a ser encontrada $T (2 \leq T \leq 15)$. As entradas não possuem espaços em branco.

Saída

Para cada caso de teste na entrada, seu programa deve imprimir uma única linha, contendo a quantidade de sequencias que se repetem dentro do segmento.

Exemplos

Entrada	Saída
AAGG AAGG AAGG 2	3

Entrada	Saída
GGGTTTCCC AAAGGGGTT CCCTTTGGG 4	0

Solução

Problema envolvendo manipulação de strings. O objetivo era receber três strings que representaria cada vírus e verificar se havia alguma substring em comum nas três strings de acordo com o tamanho informado. A saída deveria apresentar o número de substrings distintas que se repetiam nas três strings informadas.

Problema D

Bons amigos

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

Autor: Lucio Nunes de Lira (Fatec São Paulo)

Você já deve conhecer a história de João e Maria não é? Bem, esqueça, ela não é fiel ao que acontece na versão mais moderna da história onde João, Maria e a Bruxa são na verdade melhores amigos. Eles vivem em tanta harmonia com a antiga Bruxa má (atualmente Bruxa boa), que sempre participam de jogos e competem para descobrir quem é o melhor para encontrar os doces que estão escondidos na floresta.

O jogo funciona da seguinte forma, alguém fora do trio esconde os doces em algum lugar da floresta, para então João, Maria e Bruxa começarem a procurar. Para procurarem, a cada passo dado, jogam uma migalha de pão no chão. Quando um dos personagens encontra os doces, são contabilizadas quantas migalhas de pão gastou e, ao final, os valores são comparados e verifica-se quem foi mais eficiente, gastando menos migalhas. É importante ressaltar que na floresta existem muitas árvores o que pode atrapalhar ou mesmo impedir a passagem de algum dos participantes.

Os três personagens começam de um ponto específico (nunca igual ao de um amigo), um de cada vez, ou seja, enquanto um estiver na floresta procurando os doces os outros estarão aguardando em suas posições, sendo que o personagem da vez poderá inclusive passar pelos mesmos locais em que estão seus colegas. Em relação à movimentação, todos só podem andar na vertical e na horizontal, porém, existe uma regra de conduta entre os amigos, cada um deles assume que cumprirá uma ordem para a escolha das direções, sendo que a mais prioritária está mais à esquerda e a menos prioritária está mais à direita. Guie-se pela rosa dos ventos ilustrada na Figura 1

- Bruxa: *Leste, Oeste, Norte e Sul;*
- João: *Norte, Leste, Sul e Oeste*;
- Maria: *Sul, Norte, Oeste e Leste*.

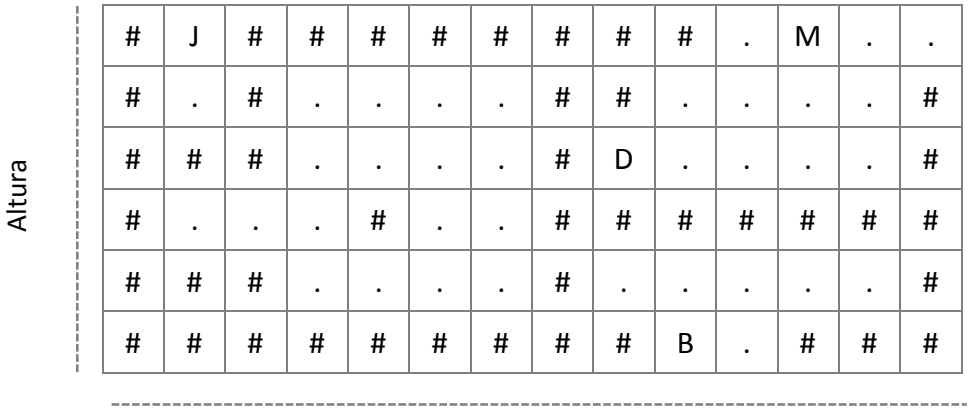
Figura 1: Rosa dos ventos.

Ou seja, sempre que for possível escolher uma direção para seguir caminho, João (por exemplo) optará em ir ao *Norte*. Caso esteja bloqueada, a direção escolhida será *Leste*; se estiver impedido, escolherá *Sul*; se nenhuma das anteriores for possível, escolherá *Oeste*.

Uma possível configuração da floresta, com a posição dos personagens, os caminhos livres, as árvores e os doces, está ilustrada na Figura 2.

É válido lembrar alguns pontos:

- Como são amigos, nenhum dos personagens tentará trapacear, portanto não podem sair dos limites da floresta, contornando-a para tentar chegar aos doces;
- Cada personagem pode começar em qualquer lugar da floresta, porém jamais na mesma posição de um colega;
- Todo personagem joga uma migalha em todo ponto que pisar, ou seja, a posição inicial e a posição final (a dos doces, se forem encontrados) também terão migalhas.



#	J	#	#	#	#	#	#	#	#	.	M	.	.
#	.	#	.	.	.	.	#	#	.	.	.	.	#
#	#	#	.	.	.	.	#	D	.	.	.	.	#
#	.	.	.	#	.	.	#	#	#	#	#	#	#
#	#	#	.	.	.	.	#	.	.	.	.	.	#
#	#	#	#	#	#	#	#	#	B	.	#	#	#

Figura 2: Possível configuração da floresta.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N $(0 \leq N \leq 100)$, indicando o número de casos. A segunda linha contém dois valores inteiros A e L $(4 \leq L, C \leq 100)$ que indicam a altura e a largura da floresta, vista como representada na Figura 2. As próximas A linhas contêm L caracteres representando a floresta, onde o caractere '#' representa uma árvore, o caractere '.' um espaço livre, o caractere 'J' representa a posição inicial de João, o caractere 'M' a posição inicial de Maria, o caractere 'B' a posição inicial da Bruxa e o caractere 'D' a posição dos doces.

Saída

Para cada caso de teste, deverá ser impresso o nome de quem foi mais eficiente, podendo ser *joao* (minúsculo e sem acentuação), *maria* (minúsculo) ou *bruxa* (minúsculo), um espaço em branco e a quantidade de migalhas gastas pelo personagem vencedor. Caso ocorra um empate entre dois ou mais personagens que conseguiram chegar aos doces, imprima apenas *empate* (minúsculo). Caso ninguém consiga chegar aos doces, imprima somente *ninguem* (minúsculo e sem acentuação). Imprima uma quebra de linha após cada caso.

Exemplos

Entrada	Saída
5 6 14 #J########.M.. #.#....##....# ###....#D....# #...#..####### ###....#.....# #########B.### 9 20 ######.BJM.######### ######.....######### ####.#.#.#.######### ####.#.#.#.######### ####.#.#.#.######### ####.#.#.#.######### ####.#.#.#.######### ####.#.#.#.######### ####..............D# 4 10 .#........ D#M.J..... .########. ....B 4 5 M#J#B .###. .###. ..D.. 6 5 ##### #BJM# ##### ##### ##D## #####	maria 9 joao 19 bruxa 12 empate ninguem

Solução

Um enunciado que assusta pelo tamanho e pelo nível de detalhes. Neste problema conceitos como uso de pilha, recursividade, manipulação de matrizes, leitura de caracteres e força bruta são exigidos.

Primeiramente, é necessário ler a quantidade de casos de teste propostos (que não era tão grande); depois armazenar o número de linhas e de colunas que serão lidas; o próximo passo é popular uma matriz, algo fácil de cumprir usando apenas dois laços de repetição.

É depois da coleta de dados da entrada que o verdadeiro desafio começa. Note que temos três personagens que devem percorrer a matriz e, de alguma forma, registrar o percurso, afinal queremos saber a quantidade de migalhas gastas. Um modo simples de evitar que áreas já verificadas sejam novamente contabilizadas é

demarcando-as, evitando assim que os mesmos caminhos sejam refeitos, culminado em um estouro de pilha, já que cada passo dado será uma chamada recursiva (entendeu o uso de "pilha"?).

Para cumprir o percurso é necessário construir funções que façam chamadas recursivas na ordem em que cada personagem se desloca, ou seja, chamadas que colocam o personagem ao norte, ao sul, ao leste e ao oeste (cima, baixo, direita e esquerda, respectivamente). Também é possível fazer isso com apenas uma função… fica como desafio para os mais destemidos.

Um importante detalhe é que cada personagem fará modificações na matriz durante seu trajeto, logo será útil construir cópias. Após o término do percurso de cada personagem, basta varrer sua matriz-cópia contando quantas migalhas gastou. Lembre-se que é vital saber se cada personagem ao menos conseguiu chegar até os doces.

Por último, é preciso verificar em qual situação o caso de teste se encontra, a dica é seguir essa ordem:

- Ninguém: mesmo com muito esforço os doces não foram encontrados;
- Bruxa: a antiga vilã conseguiu gastar menos migalhas;
- Maria: aparentemente a protagonista feminina foi a mais econômica;
- João: o jovem personagem teve a melhor estratégia e gastou pouco;
- Empate: se nenhum dos casos anteriores é verdade, só resta esse.

Problema E

Prefeito Tecnológico

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

Autor: Anderson Viçoso de Araújo (UFMS)

Estocolmo na Suécia é conhecida por ser uma das cidades mais tecnológicas do mundo. Aproveitando esse viés tecnológico da cidade, o prefeito teve uma brilhante ideia. Ele decidiu criar um aplicativo para smartphones em que a população pode votar quais as melhorias da cidade devem ter mais prioridade para serem implementadas pela administração pública. Com isso, ele espera que a população fique mais satisfeita com o seu mandato, e consequentemente, consiga se reeleger nas próximas eleições.

O aplicativo é simples. As pessoas se cadastram e vêm uma lista das possíveis melhorias que podem ser realizadas pela prefeitura na cidade. Cada pessoa pode selecionar uma única melhoria que mais lhe agrada e efetuar o voto. Depois de um período fixo de votação, o prefeito e seus assessores vão realizar as melhorias de acordo com o seu custo e quantidade de votos recebidos pelo aplicativo.

Antes de contratar o programador responsável pelo desenvolvimento do aplicativo, o prefeito sugeriu que pré-projetos com o algoritmo principal para a solução do problema fossem enviados para uma seleção inicial e você, que sonha em sair do Brasil, decidiu arriscar.

Entrada

A primeira linha de entrada contém o número de casos de teste $N (1 \leq N \leq 50)$. Cada caso de teste inicia com uma linha contendo dois inteiros, $T (1 \leq T \leq 100)$ e $L (1 \leq L \leq 100)$, separados por um espaço em branco, que representam o total (em milhões) em caixa da prefeitura para a implementação das melhorias e o número de itens da lista de melhorias, respectivamente. As L linhas seguintes, contêm um inteiro $C (1 \leq C \leq 100)$, indicando o custo total (em milhões) para implementar aquela melhoria; e um inteiro $V (1 \leq V \leq 1000)$ indicando a quantidade de votos (em milhares) que aquela melhoria recebeu, ambos separados por um espaço em branco.

Saída

Para cada caso de teste, imprima uma linha contendo o maior número de votos atendidos e a quantidade de dinheiro que vai sobrar em caixa, ambos separados por um espaço em branco. Caso não seja possível atender nenhuma melhoria imprima “NO FUNDS” (sem aspas).

Exemplos

Entrada	Saída
3 50 3 20 50 10 500 35 750 100 5 20 250 35 4 66 50 5 156 12 500 10 2 100 5 55 35	1250 5 706 17 NO FUNDS

Solução

O problema consiste em maximizar a quantidade de votos para o limite de dinheiro em caixa da prefeitura. Desta forma, o problema pode ser moldado como o problema da mochila binário (knapsack 0-1) A ideia é usar um algoritmo de programação dinâmica através de uma matriz que armazena valores para combinações de valores já conhecidas. Além disso, é necessário utilizar uma segunda matriz para armazenar o total do custo para os itens selecionados para serem implementados pelo prefeito (itens na mochila) e depois subtrair do valor *T* de entrada. Caso não seja possível atender nenhuma melhoria imprima “NO FUNDS”.

Problema F

Pastas

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

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Méridjeine é uma programadora *front-end* experiente. Ela é fascinada por tecnologias como Javascript, jQuery, Ionic, Angular, entre outras. É até difícil conversar com a menina. No Whatsapp, ao invés de falar tchau, ela manda um 'window.close()'. E por aí vai.

Em seu mais recente projeto, um disco virtual que irá fazer frente ao mega.nz (um dos únicos nos quais seus arquivos estão teoricamente seguros), ela precisou implementar uma árvore Javascript que representa a estrutura de pastas do disco. Ela sabe que existem várias bibliotecas Javascript prontas que fazem isso, mas ela não gosta de códigos de terceiros. Segundo ela, código bom é o que ela escreve. A árvore pode ter qualquer quantidade de níveis. A Figura 1 ilustra uma árvore com 3 níveis.

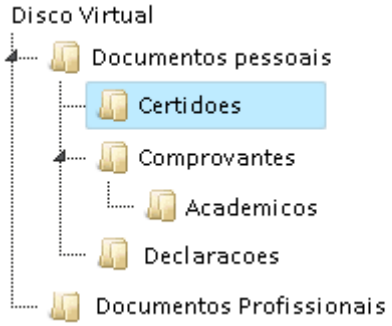


Figura 1. Exemplo de árvore de pastas com 3 níveis.

No presente momento do projeto, ela precisa implementar uma funcionalidade que pesquisa pastas por parte do nome. Para cada pasta encontrada, ela deseja imprimir o caminho completo da pasta. Para isso, ela pediu a sua ajuda.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém um número inteiro $N (1 \leq N \leq 200)$, que indica o número de pastas que serão informadas. As próximas N linhas contêm um número inteiro $I (1 \leq I \leq 200)$, um texto de tamanho máximo $T (1 \leq T \leq 50)$ e um número inteiro $P (0 \leq P \leq 200)$, separados por espaços em branco. O número I é um identificador único da pasta, o texto T é o nome da pasta e o número P é o identificador da pasta pai – isto é, o identificador da pasta dentro da qual a pasta especificada nesta linha se encontra. Um valor 0 (zero) indica a pasta raiz. Caso a pasta especificada na linha esteja localizada na raiz da árvore, o identificador P será igual a 0 (zero). A próxima linha do caso de teste contém um inteiro $M (1 \leq M \leq 10)$, que indica o número de pesquisas que serão realizadas. Por fim, as próximas M linhas contêm um texto S cada (o tamanho máximo de S é T) com um texto que deverá ser pesquisado na árvore.

Saída

Para cada pesquisa de cada caso de teste, imprima o caminho completo da pasta encontrada ou o texto "NOT FOUND" (sem aspas) se nenhuma pasta com o nome especificado tiver sido encontrada. Caso mais de uma pasta seja encontrada em uma pesquisa, imprima todas as pastas (uma por linha), assumindo a ordem alfabética. O texto digitado na pesquisa (entrada) não precisa ser exatamente igual ao nome da pasta. Basta que o nome da pasta contenha este texto para que ela seja

retornada. Desta forma, se o texto procurado for ‘Documento’, tanto a pasta ‘Documentos Pessoais’ quanto ‘Documentos Profissionais’ será um resultado válido da pesquisa. A pesquisa é *case-sensitive*.

Exemplos

Entrada

```
6
1 Documentos Pessoais 0
3 Certidoes 1
2 Documentos Profissionais 0
5 Academicos 4
6 Declaracoes 1
4 Comprovantes 1
5
Declara
Root
Doc
C
O
```

Saída

```
\Documentos Pessoais\Declaracoes
NOT FOUND
\Documentos Pessoais
\Documentos Profissionais
\Documentos Pessoais\Certidoes
\Documentos Pessoais\Comprovantes
\Documentos Pessoais
\Documentos Pessoais\Comprovantes
\Documentos Pessoais\Comprovantes\Academicos
\Documentos Profissionais
```

Solução

Estruturas de pastas de sistemas de arquivos podem ser representadas como árvores. Elas possuem um nó raiz (no caso do Windows, algo como C:) e nós filhos em diversos níveis. Existem várias soluções possíveis para o problema. A mais fácil parece ser manter dois vetores, um com o nome das pastas e o outro com o id do “pai” das pastas. A própria posição no vetor pode ser usada como id das pastas. Quando uma consulta for realizada, deve-se iterar pelos elementos do vetor de nomes e procurar aqueles que contenham o texto procurado. Ao ser encontrada alguma, o id e o nome dela podem ser adicionados à uma lista. Após identificar todas as pastas que atendem à consulta, os elementos da lista devem ser ordenados. As linguagens de programação utilizadas na maratona já possuem algoritmos de ordenação implementados. No momento da impressão, pode-se definir o caminho completo da pasta por meio de uma estrutura de repetição na qual as pastas são recuperadas até que o id da pasta “pai”, no vetor que armazena estes ids, seja igual a 0 (raiz). Outra abordagem seria utilizar structs ou classes para representar os nós da árvore e seus relacionamentos.

Problema G

E aí, professor, tem jeito?

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Pedrinho é um aluno meio atrapalhado. Com muito custo entrou na Fatec, mas está com problemas para ser aprovado: falta muito, é desorganizado, chega atrasado frequentemente, não anota informações cruciais sobre as aulas e atividades desenvolvidas ao longo do semestre, é um horror. Conseguiu ficar encrencado até em uma disciplina muito tranquila do primeiro semestre. Nessa disciplina a nota final é calculada com base no resultado de duas provas (que chamaremos aqui de P1 e P2) e na média de um conjunto de 12 atividades práticas realizadas ao longo do semestre. Cada prova e cada atividade possui uma nota entre 0.0 e 10.0. A média aritmética simples da P1 com a P2 corresponde a 80% da nota final da disciplina, ou seja, se um aluno conseguiu 5.0 na P1 e 7.0 na P2, por exemplo, esses resultados contribuem com 4.8 na nota final da disciplina. Os 20% restantes são dados pela média das 75% melhores notas das atividades, isto é, das 12 notas de atividades que o aluno tem, é calculada a média das 9 melhores. Se com esse cálculo o aluno consegue uma nota final igual ou maior a 5.75, ele é considerado aprovado (pois a nota mínima é 6.0 arredondada de meio em meio ponto). Caso o resultado do aluno seja inferior a 5.75 ele deverá fazer uma reavaliação sobre toda a matéria dada, que produzirá uma nota de 0.0 a 10.0 que será utilizada substituindo a pior nota entre a P1 e a P2 na fórmula da nota final. Se com essa substituição o novo resultado final for igual ou superior a 5.75, o aluno será considerado aprovado, senão estará reprovado.

Pedrinho fez o favor de perder a P1 e a P2 da disciplina, pois também é muito azarado e não conseguiu vir à Fatec no dia das duas provas. Mas nem tudo está perdido e ele fez algumas atividades, embora seu cachorro tenha comido algumas delas antes dele entregar. A questão que nosso amigo vagal precisa resolver agora é quanto ele precisa tirar na reavaliação para ser aprovado, se isso ainda for possível.

Entrada

A entrada consiste de vários casos de teste. Inicialmente é informado um inteiro N, $1 \leq N \leq 50$, que indica a quantidade de casos de teste a serem processados. Seguem-se então N linhas, cada uma contendo 12 valores reais V, $0.00 \leq V \leq 10.00$, com duas casas após a vírgula, separados por um espaço em branco, indicando as notas que Pedrinho obteve nas atividades práticas.

Saída

Caso Pedrinho ainda possa ser aprovado, imprimir a nota que ele precisa, na forma de um número real com uma casa decimal. Se Pedrinho não tem mais chances de ser aprovado, imprimir a mensagem "REPROVADO" em maiúsculas. Utilize números reais de precisão simples para fazer os cálculos.

Exemplos

Entrada

```
3
0.00 10.00 10.00 10.00 8.00 10.00 6.00 8.00 0.00 0.00 7.50 0.00
10.00 10.00 10.00 10.00 7.50 0.00 7.60 0.00 10.00 10.00 6.70 7.00
10.00 10.00 10.00 10.00 5.00 10.00 7.60 10.00 10.00 10.00 7.80 0.00
```

Saída

```
REPROVADO
9.8
9.5
```

Solução

O enunciado esclarece qual a fórmula de cálculo da nota final utilizada pelo professor na disciplina:

$$NF = \frac{P1 + P2}{2} * 0,8 + MedAtiv * 0,2$$

Era necessário então receber os valores das atividades e obter a média das 9 melhores notas de atividades (são 12 atividades, e 75% de 12 é 9). Isso poderia ser feito armazenando os dados lidos da entrada em um vetor e ordenando-o em sentido decrescente (e achamos a média das 9 primeiras notas) ou ascendente (nesse caso achamos a média das 9 últimas notas).

Em seguida era preciso desenvolver a fórmula do professor que determinava o valor da nota final, isolando a nota de uma prova (que será a reavaliação, que substituirá o zero de uma das provas que Pedrinho não fez), assumindo zero para a outra prova, substituindo MedAtivs pela média de atividades calculada anteriormente e igualando a equação a 5,75.

Se o valor encontrado for maior que 10, imprimir ‘REPROVADO’, senão imprimir a nota necessária com uma casa após a vírgula.

Problema H

Heap, heap!

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

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Heaps são estruturas de dados muito interessantes, que permitem solucionar alguns problemas de maneira bem elegante. Um *heap* é, antes de mais nada, uma árvore. E é uma árvore em que todos os seus níveis estão completos, com eventual exceção do seu último nível. No caso de estar incompleto, o último nível estará preenchido a partir da esquerda, como mostram as figuras 1, 2 e 3. Se em um *heap* cada nó tem o seu valor maior ou igual ao de cada um de seus filhos, ele é chamado Max-Heap; se ocorrer o inverso, ou seja, cada nó possui um valor menor ou igual ao de cada um de seus filhos, temos um Min-Heap.

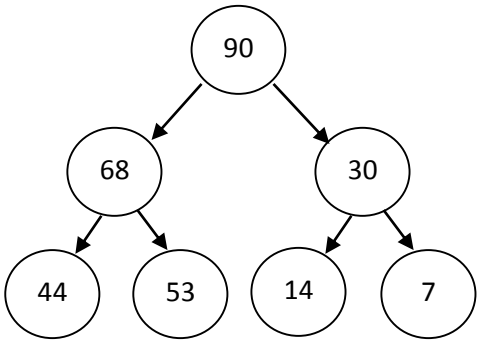


Fig 1. Um Max-Heap

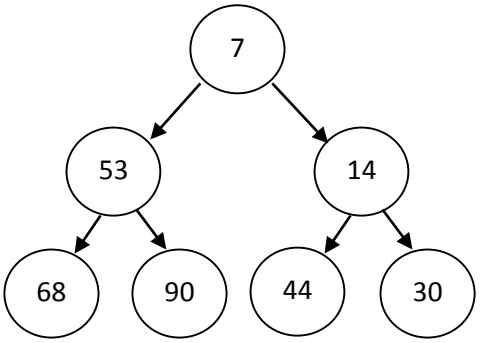


Fig 2. Um Min-Heap

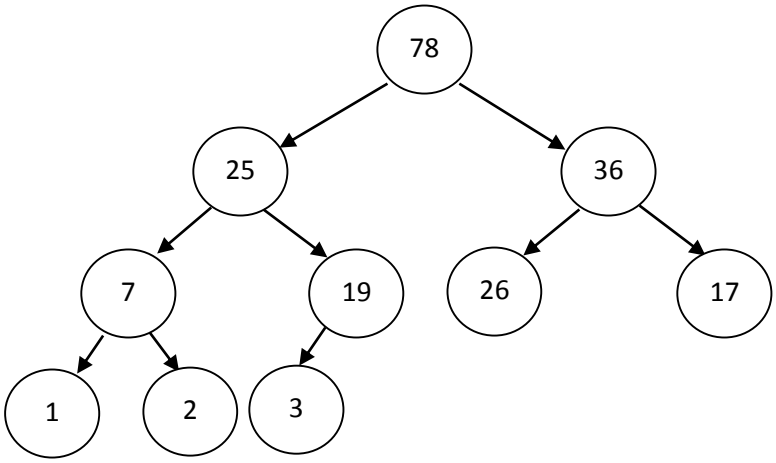


Fig 3. Um Max-Heap com seu último nível incompleto

Seu objetivo neste problema é determinar se uma árvore binária completa informada é um Max-Heap, um Min-Heap ou nenhum dos dois.

Entrada

A entrada consiste de vários casos de teste. Cada caso é dado em uma linha da entrada iniciada por um inteiro $N (2 \leq N \leq 25)$ que indica a quantidade de nós da árvore binária. Seguem-se então N inteiros V (-1000 $\leq V \leq$ 1000) correspondentes aos nós da árvore, apresentados a partir da raiz e, em cada nível, os nós são apresentados da esquerda para a direita. Considere que não serão informadas árvores com todos os seus valores iguais. O conjunto de entradas deve ser lido até a condição de fim de arquivo ser atingida.

Saída

Caso a árvore lida em um caso de teste seja um Max-Heap, imprimir a string ‘max’ (em minúsculas); caso seja um Min-Heap, imprimir ‘min’ (em minúsculas) e caso a árvore lida não seja nem um Max-Heap nem um Min-Heap, imprimir ‘nada’ (em minúsculas).

Exemplos

Entrada	**Saída**
7 90 68 30 44 53 14 7 7 7 53 14 68 90 44 30 10 78 25 36 7 19 26 17 1 2 3 7 90 53 30 44 68 14 7 7 7 53 44 68 90 14 30 10 78 19 36 7 25 26 17 1 2 3	max min max nada nada nada

Solução

Como todo heap é uma árvore completa, ele pode ser armazenado em um vetor, o que simplifica muito o processamento. Uma árvore completa tem seus elementos armazenados no vetor na ordem de seus níveis, em geral da esquerda para a direita. Isso quer dizer que a raiz da árvore (nível 0) fica no primeiro elemento do vetor, e os nós do nível 1 ficam nos elementos 1 e 2 (o problema restringia-se a árvores binárias). Os elementos do nível 2 ocupam os elementos 3, 4, 5 e 6, e assim por diante. A quantidade máxima de elementos em um nível é sempre igual ao dobro da quantidade de elementos do nível anterior, ou seja, o nível N tem 2^N elementos.

Portanto a árvore da figura 3 poderia ser armazenada em um vetor da seguinte forma:

78	25	36	7	19	26	17	1	2	3					

Para determinar se tratava-se de um max-heap, era necessário percorrer, para cada elemento, a posição dos seus descendentes e verificar se todos eram menores ou iguais ao elemento raiz daquela subárvore. Se para algum elemento isso não fosse verdade, aquela árvore não era um max-heap. O mesmo raciocínio vale para o min-heap, apenas mudando o operador de comparação, pois nesse caso nenhum descendente pode ser menor que o seu ancestral direto.

Problem I

Minimum Wage

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

Author: Danilo Ruy Gomes (Fatec Itapetininga)

In times of financial crisis, one way out is to leave Brazil and work in another country. But is it really worth living and working abroad? This month, news articles have been published about the considerable increase in the cost of living in developed countries.

Pedrinho, a developer who graduated at FATEC about two years ago, with experience in programming, is facing this dilemma. For this reason, he decided to take into account some items he considers important, such as housing, food and transportation. The earnings not destined to such items would be invested in a fund that yields about 1% per month. For example, he worked for three months in a company in Sao Paulo (Brazil) and received R$ 120.00 per hour. Taking into consideration a journey of 160 hours per month, the percentage of expenditure with housing, food, and transportation never exceeded 70% of his earnings. The remainder was applied in a fund that yielded 1% per month. After three months, he had an amount of R$ 17,627.91. If he had lived in England, we know that the amount spent with food and transportation would decrease, but expenses with housing would increase. However, as the value of a Pound is about 4.60 reais, it could be worth.

Based on this idea, and having some parameters such as currency exchange rate, earnings per hour, number of working hours, and percentage of earnings spent with housing, food and transportation, help our friend Pedrinho to decide if it is already time to move to another country or to stay in Brazil. To answer that, compare the balance of his account in Brazil with the balance in the foreign country. Consider that he works for 3 months and applies his earnings (after paying expenses with housing, food, and transportation) in funds that yield 1% per month.

Input

The input consists of a single test case. The first line of the test case contains his earnings per hour in Brazil HBV ($10.00 \leq HBV <400.00$), the earnings per hour in the foreign country HEV ($10.00 \leq HEV \leq 200.00$), the number of working hours in a month QH ($1.00 \leq QH \leq 200.00$), the percentage of earnings spent with housing, food and transportation in Brazil DPB ($20.00 \leq DPB <90.00$), the percentage of such expenses in the foreign country DPE ($20.00 \leq DPE <90.00$), and the currency exchange rate CT ($1.00 \leq CT \leq 20:00$). All numbers have two decimal places. Use floating point numbers with double precision throughout the processing.

Output

The output of your program must write the amount money he will have after three months to two decimal places, followed by BR for Brazil and ES for the foreign country.

Examples

Input	Output
`120.00 27.00 160.00 70.00 71.00 4.60`	`17627.91BR 17636.72ES`

Input	Output
`100.00 31.00 120.00 70.00 85.00 4.03`	`11017.44BR 6882.05ES`

Solução

Problema simples envolvendo calculo com juros compostos. A ideia era informar sua arrecadação mensal e suas despesas, assim como o câmbio do pais estrangeiro. A saída deveria ser calculada com a sobra do salário utilizando-se juros compostos com valor fixo de 1% durante 3 meses, tanto no Brasil como no estrangeiro.

A soma deveria levar em consideração a sobra de cada mês mais os juros do mês anterior e todos os valores deveriam ser expressos como dados do tipo real de precisão dupla, para evitar problemas de arredondamento.

Problema J

Sushiman Matemático

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

Autor: Anderson Viçoso de Araújo (UFMS)

Kyoshi não é só um simples sushiman, ele é formado em matemática pela UFMS. Ele sempre gostou de gastronomia e depois de um tempo de formado, conseguiu se dedicar a um curso de sushiman que estava querendo fazer a muito tempo. Os amigos adoraram a ideia de ter um sushiman para cozinhar para eles e Kyoshi também gosta muito de receber os amigos.

O sushiman Kyoshi sempre produz um excelente sushi, mas como bom matemático, tem as suas manias. Você pode perceber isso olhando os dígitos amorosamente estampados em papel comestível que ele coloca em cada rolinho de sushi produzido por ele. O sushiman garante que um degustador de suas delícias recebe sempre um rolinho de sushi com um número primo. Além disso, quando cortado pela direita, os números sobre cada corte continuam primos até o último pedaço cortado, por exemplo: O sushi com número 7331, que é primo; os três pedaços 733 representam um primo; os outros dois com o número 73 também representam um primo; e, é claro, o último pedaço, 7. O número 7331 é chamado de Super Primo de comprimento 4.

Seu trabalho é ajudar Kyoshi a criar um programinha que aceita um número de sushis como entrada e imprime todos os super primos desse comprimento. Lembrando que o número 1 (por si só) não é um número primo.

Entrada

A entrada é composta por uma linha com um número $N (0 < N < 9)$.

Saída

Imprimir os números super primos de comprimento N em ordem ascendente, um por linha.

Exemplos

Entrada	**Saída**
4	2333 2339 2393 2399 2939 3119 3137 3733 3739 3793 3797 5939 7193 7331 7333 7393

Solução

A solução para esse problema pode ser considerada relativamente simples. A ideia é armazenar todos os super primos de tamanho 1 até 8 antes de efetuar as consultas e não durante. Com isso, o tempo é bastante reduzido a cada consulta sobre os primos de um certo tamanho. Basta começar com os primos de tamanho 1 (2, 3, 5 e 7) e depois ir verificando novos primos de tamanhos maiores a partir deles.

Uma outra solução é implementar um programa que gere todos os super primos e armazená-los em um vetor (ou mais vetores, por tamanho por exemplo) de constantes no código-fonte. Depois, para cada consulta, verificar se o número se encontra no vetor.