

Universidade Federal de Goiás – UFG
Instituto de Informática – INF
Bacharelados (Núcleo Básico Comum)

Algoritmos e Estruturas de Dados 1 – 2021/2

Lista de Exercícios nº 04 – Listas Lineares, Filas e Pilhas
Turmas: INF0286/INF0061 – Prof. Wanderley de Souza Alencar

Sumário

1	A COPA do mundo é nossa...	2
2	A lista de Arya (série “Game of Thrones”)	5
3	Micalateia	8
4	Brincadeira	10
5	Fila do SUS	12
6	Branca de neve e os n anões	15
7	<i>Programming strategy</i>	20
8	Balanceamento de parênteses	23
9	Caminho	26
10	Notação polonesa inversa	30
11	Estação de trem	32
12	Expressões	35

Pontuação

Para obter nota 10 nesta lista é necessário resolver um conjunto de exercícios cujos pontos somados correspondam a 12 pontos, no mínimo. Os alunos podem escolher qualquer combinação possível de exercícios, tal que a soma acumulada atinja 12 pontos. Entretanto, incentiva-se fortemente que sejam resolvidos o maior número de exercícios possíveis (mesmo que extrapole a quantidade necessária para atingir os 12 pontos).



1 A COPA do mundo é nossa...



(+)

Sempre que se aproxima a realização de uma *Copa do Mundo de Futebol*, o fluxo de pessoas nas filas para compra de ingressos aumenta consideravelmente. Quando as filas vão se tornando cada vez maiores, as pessoas menos pacientes tendem a desistir da compra de ingressos e acabam deixando as filas, liberando, assim, sua vaga para outras pessoas, o que é muito bom para os mais pacientes.

Quando uma pessoa deixa a fila, todas as pessoas que estavam atrás dela dão um passo à frente e, portanto, nunca existe um espaço vago entre duas pessoas consecutivas na fila. A fila inicialmente contém n pessoas, cada uma com um identificador diferente: m_i . Estes identificadores são, vulgarmente, chamados de “*senhas*”.

Pedro, um menino que sagrou-se campeão da etapa regional goiana da OBI (Olimpíada Brasileira de Informática, coordenada pelo Prof. Wellington Martins do INF/UFG), sabe o *estado inicial* da fila e a sequência de identificadores das pessoas que deixaram a fila, na ordem em que elas a abandonaram. Ele também sabe que, após o *estado inicial*, nenhuma pessoa entrou mais na fila.

Ele deseja saber qual será o *estado final* desta fila, mas não quer fazer, manualmente, a simulação das saídas das pessoas. Pedro não conhece “Estruturas de Dados” e, sabendo que você está cursando esta disciplina no INF/UFG, pediu sua ajuda para elaborar um programa de computador, \mathbb{C} , que resolva este problema.

Entrada

A primeira linha contém um número natural n representando a quantidade de pessoas inicialmente na fila, com $1 \leq n \leq 60000$.

A segunda linha contém n números naturais representando os identificadores m_i das pessoas na fila, com $1 \leq i, m_i \leq n$. O primeiro identificador corresponde ao identificador da primeira pessoa na fila, o segundo identificador é o da segunda pessoa na fila e, assim, sucessivamente. É garantido que duas pessoas diferentes não possuem o mesmo identificador.

A terceira linha contém um natural s , $1 \leq s \leq n$, que representa a quantidade de pessoas que deixaram a fila. A quarta linha contém s naturais – s_i , $1 \leq s_i \leq n$, representando os identificadores das pessoas que deixaram a fila, na ordem em que elas a abandonaram. Da mesma maneira, é garantido que um mesmo identificador não aparece duas vezes nessa lista de pessoas que saíram a fila.

Saída

Seu programa deve imprimir uma única linha contendo $(n - m)$ naturais, correspondendo à sequência de identificadores das pessoas que *permaneceram* na fila, na ordem de chegada a ela.

Exemplos

Entrada	Saída
8 5 100 9 81 70 33 2 1000 3 9 33 5	100 81 70 2 1000

Entrada	Saída
4 10 9 6 3 1 3	10 9 6

Entrada	Saída
8 10 2 3 4 66 45 32 77 1 3 10 2 3	4 66 45 32 77

Entrada	Saída
8 10 2 3 4 66 45 32 77 1 5 10 3 45 32 77	2 4 66

Entrada	Saída
16 7 1 3 8 2 13 9 6 4 14 10 5 15 11 12 16 5 11 10 2 7 16	1 3 8 13 9 6 4 14 5 15 12



2 A lista de Arya (série “Game of Thrones”)



(+)

Lista de Arya: “Cersei. Walder Frey. Montanha. Meryn Trant.”

Para se manter motivada, Arya sempre lembra a lista de inimigos que ela mais odeia. O principal objetivo de sua jornada é acabar com TODOS na sua lista!

Entretanto, às vezes algum inimigo dela pode ser morto por outra pessoa – o que é lamentável. Quando ela descobre que tal inimigo morreu, ela o remove da sua lista. Além disso, Arya também pode fazer novos inimigos durante sua jornada. Quando ela faz um novo inimigo, tal inimigo é incluído na sua lista.

Arya quer acabar com seus inimigos um por um, na mesma ordem em que aparecem na sua lista. A qualquer momento, ela pode se perguntar quanto tempo irá levar para acabar com todos que estão entre dados dois inimigos. Para tal, dados dois inimigos a e b , ela deve determinar quantos inimigos estão na lista entre a e b , excluindo ambos. Ajude Arya respondendo tais perguntas.

Entrada

A primeira linha contém um inteiro n ($1 \leq n \leq 1000$), o número de inimigos inicialmente em sua lista. Considere que todas as pessoas *inimigas* são numeradas de 1 a 1000, inclusive extremos.

A próxima linha contém n inteiros, descrevendo a lista inicial de Arya. As linhas seguintes descrevem as operações que podem ser realizadas na lista de Arya. Cada operação pode estar em um dos seguintes formatos:

- I p e** Insira a pessoa de número p depois do inimigo de número e na lista. É garantido que e está na lista, e que p não está na lista, com $(1 \leq e, p \leq 1000)$. Deve imprimir, como saída, a expressão: “inserido ” p .
- R p** Remova o inimigo de número p da lista. É garantido que p está na lista. Deve imprimir, como saída, a expressão: “removido ” p .
- C a b** Determine a quantidade de inimigos estão na lista entre os inimigos de números a e b , excluindo ambos. É garantido que a e b estão na lista, mas não se sabe a ordem em que eles estão. Deve imprimir, como saída, a expressão: “quantidade ” x , onde x é o número de inimigos.

- M** Mostre os números os inimigos que estão na lista, a partir do primeiro em direção ao último. Deve imprimir, como saída, a expressão: “lista ” $p_1 p_2 p_3 \dots p_n$, considerando que há n inimigos na lista de Arya. Se a lista estiver vazia, deve-se mostrar a mensagem “lista vazia”.
- F** Termina as operações com a lista de Arya e, por consequência, a execução do programa, imprimindo a palavra “fim”.

Saída

Imprima uma linha por operação realizada, conforme mostram os exemplos a seguir.

Exemplos

Entrada	Saída
3 3 8 2 C 3 2 I 9 8 C 3 2 R 8 I 1 2 C 1 9 F	quantidade 1 inserido 9 quantidade 2 removido 8 inserido 1 quantidade 1 fim

Entrada	Saída
10 7 5 1 10 2 4 6 9 3 8 I 11 10 I 14 7 I 12 9 R 3 R 6 R 2 C 7 5 C 11 12 C 10 8 C 7 8 C 1 5 M F	inserido 11 inserido 14 inserido 12 removido 3 removido 6 removido 2 quantidade 1 quantidade 2 qunatidade 4 quantidade 8 quantidade 0 lista 7 14 5 1 10 11 4 9 12 8 fim

Entrada	Saída
10 7 5 1 10 2 4 6 9 3 8 C 7 5 C 10 3 C 9 8 C 1 8 F	quantidade 0 quantidade 4 quantidade 1 quantidade 6 fim

Entrada	Saída
2 1 10 I 9 1 I 7 9 I 8 1 C 1 10 C 8 7 C 9 7 R 1 R 10 C 8 7 F	inserido 9 inserido 7 inserido 8 quantidade 3 quantidade 1 quantidade 0 removido 1 removido 10 quantidade 1 fim



3 Micalateia



(+)

A cidade de Pentescopéia é muito atrasada (ainda usam telefones da década de 1980!), e somente três pessoas possuíam telefone, uma delas era Micalateia, irmã de Hermanoteu. Micalateia possui um *hobbie* muito esquisito: ela anota em sua agenda de contatos o número de vezes em que ela ligou para uma pessoa. A agenda de Micalateia registra, até o momento, o seguinte:

- Hermanoteu 4523-2248 300 ligações
- Ooloneia 4523-4887 299 ligações

Como ultimamente tem aumentado o número de pessoas com telefone em Pentescopéia, sua agenda está crescendo e ela pediu para que você criasse um programa, em \mathbb{C} , que a permitisse realizar as seguintes funções:

- Inserir um novo contato;
- Remover um contato existente na lista;
- Registrar quem, da lista de contatos existente, fez uma ligação;
- E que essa lista seja ordenada por quem ela ligou mais vezes.

Entrada

Cada linha do arquivo de entrada pode possuir algum dos seguintes formatos:

I nome tel v

Insira a pessoa com `nome` (de até 20 caracteres), telefone `tel`, e `v`, correspondendo ao número de vezes que ela (Micalateia) ligou para esta pessoa;

R nome

Remova a pessoa que possui nome igual a `nome` da lista;

L nome

Aumenta, em uma unidade, o número de ligações que Micalateia fez para a pessoa cujo nome é `nome`;

F

Termina aquela sessão de operações na agenda.

Observações:

- É garantido que `nome` está na lista quando ocorre uma operação de remoção, como também não se insere alguém que já esteja na lista;
- Na lista nunca haverá duas pessoas com nomes idênticos, como também não haverá ninguém com nome composto;
- Sempre que o número de ligações for igual, o primeiro que possuir esse número fica na frente na lista.

Saída

A listagem de saída deverá conter uma pessoa por linha, na ordem em que estão dispostas a partir do início da lista na agenda, obedecendo ao seguinte formato:

- nome da pessoa;
- símbolo de hífen, precedido e sucedido por um único espaço em branco;
- número de telefone no formato `xxxx-xxxx`, onde cada “x” representa um dígito;
- número de ligações realizadas, precedidas por um único espaço em branco.

Exemplos

Entrada	Saída
L Ooloneia I Dirineia 4523-6667 0 I Mirineu 4523-1313 1 L Dirineia L Dirineia F	Hermanoteu - 4523-2248 300 Ooloneia - 4523-4887 300 Dirineia - 4523-6667 2 Mirineu - 4523-1313 1

Entrada	Saída
I Casimiro 7777-7777 10 I Zelia 8888-8888 100 I Machado 9999-9999 63 I Bilac 6666-6666 89 L Bilac L Bilac L Machado L Zelia L Casimiro F	Zelia - 8888-8888 101 Bilac - 6666-6666 91 Machado - 9999-9999 64 Casimiro - 7777-7777 11



4 Brincadeira



(+)

Para acalmar seus cinco netos, Dona Arlete propôs uma brincadeira:

Um neto era *vendado* e o restante se agrupava formando um círculo, sendo que a vovó ocupava a posição de número “um” nesse círculo. O neto *vendado* dizia um nome (de outro dos netos) e uma direção: horário ou anti-horário.

Se em até dois passos, sem contar a vovó, ele acertasse o nome de um dos outros netos, então marcava um ponto e o primo (ou irmão) cujo nome foi dito deixava o círculo. Esse processo se repetia para cada neto presente no círculo. Não se podia repetir os nomes.

A vovó pediu seu auxílio: é a vez de Paulo ficar vendado e, dada a disposição inicial dos outros netos no círculo, bem como os nomes e direções ditas por Paulo, determine a pontuação feita por ele.

Você deverá escrever um programa \mathbb{C} para realizar esta tarefa, segundo os padrões estabelecidos a seguir.

Entrada

As primeiras linhas contém, cada uma, o nome de um dos netos de dona Arlete, do primeiro do círculo ao último, inseridos sempre no sentido horário. A leitura de nomes deve ser encerrada pela digitação da palavra FIM, com todas as letras maiúsculas.

Na sequência, há uma linha por neto no círculo, onde está o nome de um neto e a direção (horário ou anti-horário).

Observação: Lembre-se que a Dona Arlete, a vovó, é que inicia o círculo ocupando sempre a posição de número “UM” e que, na entrada, as palavras “horário” e “anti-horário” são escritas sem a acentuação gráfica.

Saída

Imprima a pontuação de Paulo.

Entrada	Saída
Ana Joaquim Henrique Marcela Carlos Sabrina Loys FIM Henrique horario Ana horario Joaquim horario Marcela horario Carlos anti-horario Sabrina anti-horario Loys anti-horario	5

Entrada	Saída
Ana Joaquim Henrique Marcela Carlos Sabrina Loys FIM Loys anti-horario Sabrina anti-horario Henrique horario Marcela horario Ana horario Joaquim horario Carlos anti-horario	5

Exemplos

Atendimento pelo SUS



5 Fila do SUS



(++)

Os pacientes que chegam à fila do *Sistema Único de Saúde* (SUS) passam por uma *triagem* e, imediatamente, vão para a fila de atendimento ¹.

No processo de triagem, um(a) enfermeiro(a) anota o horário de entrada do(a) paciente e quantos minutos ele(a) tem até que sua condição de saúde se torne crítica, ou seja, momento a partir do qual sua morte será inevitável. Uma característica *especial* é que o(a) enfermeiro(a) **nunca erra em seu diagnóstico** e, portanto, consegue prever, de maneira inequívoca, em quanto tempo o paciente atingirá este estado crítico. Além disso faz esta atividade de maneira instantânea, ou seja, a identificação do estado de um(a) paciente não consome nenhum tempo.

Sabe-se também que os(as) pacientes são atendidos(as) de 30 em 30 minutos, com pontualidade britânica e com atendimentos sempre sendo iniciados em *horas cheias* ou *meias-horas*: 07h, 07h30min, 08h e, assim, sucessivamente. O início da triagem e do atendimento se dá, pontualmente, às 07h de cada dia.

Se não há nenhum(a) paciente sendo atendido(a) e a fila está vazia – Isto ocorre? Eu não acredito – o(a) primeiro(a) paciente é atendido(a) no instante que chega à triagem. O(A) médico(a) sempre atende até o(a) último(a) paciente na fila.

A preocupação é se algum(a) paciente atingiu a sua *condição crítica* enquanto está na fila de atendimento, ou seja, ainda não tenha sido iniciado seu atendimento.

Diante deste cenário, você foi convidado(a) a implementar um programa de computador, em \mathbb{C} , que seja capaz de verificar na fila quantos pacientes atingem a sua condição crítica.

Entrada

A primeira linha da entrada contém um número inteiro $n \in \mathbb{N}^*$, $0 < n \leq 50$, o número de pacientes que chegaram à *triagem*.

As n linhas seguintes possuem, cada uma, uma tríade de valores inteiros: $h, m, c \in \mathbb{N}^*$ e $7 < h < 19$, $0 \leq m < 60$ e $0 \leq c \leq 720$. Os valores de h e m correspondem, respectivamente, à hora e minuto em que o(a) paciente chega à *triagem*. O(A) paciente da linha i sempre chega antes de, e no máximo junto com, o(a) paciente da linha $(i + 1)$. O valor c é o tempo, expresso em minutos, antes daquele(a) paciente atingir a condição crítica em seu estado de saúde.

Saída

O programa deverá imprimir o número de pacientes que atingiram a condição crítica ainda na fila para atendimento.

Exemplos

Entrada	Saída
4 7 0 20 7 0 30 7 30 20 8 15 30	1

¹Lembre-se: Isto é apenas um exercício de programação e, portanto, não há nenhuma fidedignidade com os acontecimentos reais no SUS do Brasil.

Entrada	Saída
5 10 20 50 10 30 30 11 10 20 12 0 0 12 10 30	0

Entrada	Saída
24 7 0 0 7 0 30 7 0 60 7 0 90 7 0 120 7 0 150 7 0 180 7 0 210 7 0 240 7 0 270 7 0 300 7 0 330 7 0 360 7 0 390 7 0 420 7 0 450 7 0 480 7 0 510 7 0 540 7 0 570 7 0 600 7 0 630 7 0 660 7 0 690	0

Entrada	Saída
15 7 0 0 7 0 20 7 30 50 7 50 30 8 0 120 8 1 75 8 30 90 9 59 30 11 0 240 11 15 0 11 30 300 14 0 0 14 40 5 15 15 5 17 50 5	8

Entrada	Saída
10 7 0 0 7 30 20 7 30 50 7 50 30 8 0 90 8 10 75 8 30 60 9 50 30 11 0 140 11 15 60	4



6 Branca de neve e os n anões



(++)

Branca de Neve e os n anões vivem na floresta. Todas as manhãs, os anões formam uma longa fila indiana e vão assobiando para a mina. Branca de Neve corre ao redor deles e faz um monte de fotografias digitais – ela é uma princesa tecnológica, que acessa e atualiza diariamente a sua rede social favorita fazendo o *upload* das fotos que considera bonitas.

Quando os anões entram na mina, Branca de Neve volta para casa e passa a selecionar as fotos mais bonitas. Cada anão tem uma touca colorida, e há c cores diferentes disponíveis pela eles. Ela tem um gosto pitoresco: para ela, uma *foto bonita* é aquela em que mais da metade das toucas dos anões são da mesma cor. Explicase: se houver k anões numa foto, ela é uma *foto bonita* se estritamente mais que $k/2$ anões usam toucas de mesma cor. Os anões não sabem disso e, por isso, não são capazes de escolher as suas toucas com a intenção de, deliberadamente, agradar Branca de Neve. Cada um faz sua escolha de maneira totalmente pessoal e desconhecida dos demais.

Você, um discípulo de Malba Tahan², quer verificar, para um conjunto de m fotos, quantas são *fotos bonitas* e qual a cor predominante nas toucas.

Como você não quer fazer isto manualmente, resolveu escrever um programa de computador, em \mathbb{C} , para cumprir a tarefa.

Entrada

A primeira linha da entrada conterá um número natural t , o número de casos de teste, que virão na sequência, sabendo que $1 \leq t \leq 1000$ e que cada caso de teste possui $(3 + m)$ linhas.

A primeira linha de cada caso de teste conterá os dois naturais n e c , conforme anteriormente especificado e sabendo-se que $3 \leq n \leq 3 \times 10000$. A linha seguinte conterá n naturais: $c_1, c_2, c_3, \dots, c_n$, cada um correspondendo à cor da touca do respectivo anão, ordenados segundo a fila que eles formaram naquela manhã. É claro que $1 \leq c_i \leq c \leq n$.

A terceira linha de cada caso de teste conterá o valor de m , com $1 \leq m \leq 1000$.

As m linhas seguintes conterão, cada uma, dois naturais a e b , de tal maneira que $1 \leq a \leq b \leq n$. Cada uma destas linhas descreve uma foto, a qual contém todos os anões a partir do a -ésimo até b -ésimo, ou seja, não é obrigatório que todos os anões apareçam em todas as fotos registradas pela manhã.

²Veja em https://pt.wikipedia.org/wiki/Malba_Tahan.

Saída

A saída deverá conter, por caso de teste, m mensagens: uma por foto registrada para aquele caso de teste.

Se Branca de Neve a considerou a foto uma *foto bonita*, a mensagem deve ser “bonita X”, onde X é a cor predominante nas toucas dos anões naquela foto. Lembre-se que, em nosso caso, a cor é representada por um número natural. Do contrário, a mensagem deve ser “feia”.

Exemplos

Entrada	Saída
1	feia
10 3	bonita 1
1 2 1 2 1 2 3 2 3 3	feia
8	bonita 1
1 2	feia
1 3	bonita 2
1 4	feia
1 5	bonita 3
2 5	
2 6	
6 9	
7 10	

Entrada	Saída
2	feia
10 3	bonita 1
1 2 1 2 1 2 3 2 3 3	feia
8	bonita 1
1 2	feia
1 3	bonita 2
1 4	feia
1 5	bonita 3
2 5	feia
2 6	feia
6 9	feia
7 10	feia
12 5	feia
1 5 2 3 2 4 2 5 3 4 2 4	feia
6	
1 5	
2 7	
9 12	
3 12	
4 9	
1 12	

Entrada	Saída
1	feia
15 4	bonita 2
1 2 2 3 4 4 2 1 1 2 4 2 3 3 2	feia
8	feia
1 2	feia
1 3	feia
1 4	feia
1 5	feia
2 5	
2 6	
6 9	
1 15	

Entrada	Saída
2	bonita 1
10 4	bonita 1
1 1 1 1 1 1 1 1 1 1	bonita 1
10	bonita 1
1 2	bonita 1
1 3	bonita 1
1 4	bonita 1
1 5	bonita 1
1 6	bonita 1
1 7	bonita 1
1 8	feia
1 9	feia
1 10	feia
2 10	feia
10 4	feia
1 2 3 4 1 2 3 4 1 2	feia
10	feia
1 2	feia
1 3	feia
1 4	feia
1 5	
1 6	
1 7	
1 8	
1 9	
1 10	
2 10	

Entrada	Saída
3	feia
5 5	feia
1 2 3 4 5	bonita 2
2	bonita 2
1 3	feia
1 5	feia
3 2	feia
2 2 2	feia
2	feia
1 2	
1 3	
6 3	
1 2 3 1 2 3	
5	
1 2	
1 3	
1 4	
1 5	
2 5	



7 *Programming strategy*



(++)

The friends John, Jack and Joseph, like to participate of programming competitions. They have different strategies we would like to know which strategy is best:

John – simply solves the problems in the order in which he receives them from the contest organizers;

Jack – first reads all problems and solves them in increasing order of difficulty;

Joseph – also he reads all problems first, but he is very ambitious and, therefore, solves the problems in decreasing order of difficulty.

The *difficulty* of a problem is measured in “how many minutes” the boys take to solve the problem.

We gathered statistics and consulted the “Thierson Couto Oracle” – an oracle as famous in INF/UFG as that of *Oracle of Delphi*, in Ancient Greece. So we know, for all types of problems, how much time the boys will need.

We also found that for each problem, the three guys always need the same time, which depends on the difficulty of the problem. Thus, they differ just for their particular strategies.

For various competitions, we would like you to tell us the “winner”, the number of problems solved and what your “score”.

The “score” for a single problem is the time, in minutes, from the start of the contest until its resolution.

The *overall scoreboard* is the sum of the scores of the problems solved.

The boys never make mistakes, so you do not have to deal with penalties. The “winner” is that that solve more problems and, in case of a tie, the one with the lowest score. If there is still tie, the three boys agree that Joseph is the winner because he always brings delicious apple pies to everyone during workouts.

Input

The first line of the entry will contain an integer t , the number of test cases, where $1 \leq t \leq 1000$.

Each test case describes a contest and its first line tells how long the competition lasts d , in minutes, where $30 \leq d \leq 1440$, and number of problems p , from 3 to 24.

In a second line you will have the “difficulties of problems”, such as explained above, they say how many minutes (from 1 to 600) the guys need to solve the problems.

Output

The output must contain, for each test case, a line containing the name of the “winner”, the number of problems that he solves and, finally, his score.

Use the exact format as shown below in the example, even if the winner only solves 0 (zero) or 1 (one) problem.

Example


Entrada	Saída
2 180 6 23 42 170 33 7 19 60 2 43 17	Jack 5 288 Jack 2 77

Entrada	Saída
2 500 8 10 30 40 20 50 70 90 100 600 5 100 70 50 10 30	Jack 8 1290 Jack 5 560

Entrada	Saída
3 500 5 40 50 90 30 80 700 9 30 30 50 80 70 20 10 80 10 1000 5 200 150 50 300 250	Jack 5 710 Jack 9 1290 Jack 5 2250

Entrada	Saída
2 1000 3 400 200 100 1000 3 100 200 400	Jack 3 1100 Jack 3 1100

Entrada	Saída
3 1000 2 500 500 1000 4 250 250 250 250 1000 7 100 100 100 100 100 100 100	Jack 2 1500 Jack 4 2500 Jack 7 2800

Check for balanced parentheses		
Expression	Balanced?	() or {} or []
) (NO	
[(]	Yes	
[(])	NO	
[() ()]		
 \Rightarrow Last unclosed, first closed		

mycodeschool.com

8 Balanceamento de parênteses



(++)

Considere que seja dada uma expressão aritmética e qualquer, onde nela pode haver a presença de parênteses, ou seja, “(” – abre parêntese ou “)” – fecha parêntese.

Por exemplo:

$$a * b - (2 + c)$$

é uma expressão correta, pois há um “abre” parêntese e um “fecha” parêntese.

O mesmo ocorre com:

$$(a + b * (2 - c) - 2 + a) * 2$$

Por outro lado, a expressão:

$$(a * b - (2 + c)$$

está incorreta.

Como também estão:

$$2 * (3 - a))$$

e

$$)3 + b * (2 - c)($$

Sintetizando: todo parêntese que “fecha” deve ter um outro parêntese que “abre” correspondente, bem como não pode haver parêntese que “fecha” sem um prévio parênteses que “abre”. A quantidade total de parênteses que “abre” e “fecha” deve ser igual.

Você deve elaborar um programa de computador, em \mathbb{C} , que seja capaz de verificar se uma dada expressão e , fornecida como entrada, está correta no que diz respeito à parentetização.

Observação: É obrigatório que seu programa utilize uma estrutura de dados do tipo “dinâmica” para implementar a solução para este problema.

Entrada

Uma única expressão e , com no mínimo 1 (um) caractere e no máximo 1000 (mil) caracteres, na linha de entrada.

Saída

Seu programa deve imprimir, numa única linha, a mensagem “correta” ou “incorreta”. A análise deve ser realizada sem considerar o restante da expressão, apenas a relação existente entre os parênteses, ou seja, desconsideram-se todos os demais elementos (numerais, espaços em branco e símbolos de operação aritmética).

Exemplos

Entrada	Saída
$a * b - (2 + c)$	correta

Entrada	Saída
$) 3 + b * (2 - c) ($	incorreta

Entrada	Saída
$(a + b * (2 - c) - 2 + a) * 2 ($	correta

Observação: Perceba todas as letras das palavras *correta* e *incorreta* estão grafadas em minúscula. Você deve fazer com que seu programa grafê *exatamente* desta maneira.

Entrada	Saída
$2 \star (3 - a))$	incorreta



9 Caminho



(++)

A família de Luna, uma menina de dez anos, acaba de se mudar para uma nova cidade, e hoje é o seu primeiro dia de aulas na sua “nova” escola.

Como ela é uma menina muito distraída, sua mãe fez uma *lista de instruções* para que ela saiba, sem erro, andar de sua casa até a escola.

Cada instrução descreve uma curva que ela deve fazer para contornar uma esquina da cidade.

Por exemplo, a lista:

INSTRUÇÕES
DIREITA
QUINZE
DIREITA
FRAMBOESA
DIREITA
ESCOLA

Ela significa que Luna deve “virar à direita” na rua QUINZE, em seguida deve “virar à direita” na rua FRAMBOESA e, finalmente, “virar à direita” para chegar à ESCOLA.

Você foi incumbido de fornecer as instruções para que Luna faça o caminho contrário ao que recebeu para chegar ao seu destino. Para isto você deve elaborar um programa de computador, escrito em \mathbb{C} , para realizar a tarefa de *geração* das instruções a serem fornecidas para Luna em seu retorno para casa.

Você pode assumir que a lista que Luna recebe contém pelo menos duas e, no máximo, dez instruções. Pode também pressupor que cada linha contém, no máximo, 20 caracteres que sempre são grafados utilizando letras maiúsculas e que a última instrução sempre é para “virar” para a ESCOLA.

Entrada

A primeira linha da entrada contém um número natural t , indicando o número de casos de teste. Sabe-se $1 \leq t \leq 10$.

Para cada caso de teste há uma sequência de pares de linhas com a lista de instruções para o caminho de

casa até a ESCOLA. A primeira linha do par é uma instrução de virar à *direita* – DIREITA – ou à *esquerda* – ESQUERDA. A segunda linha é o nome da rua, exceto na última linha de cada caso de teste que, sempre, conterá a palavra ESCOLA.

Saída

Seu programa deve imprimir, na saída padrão, e para cada caso de teste, uma sequência de linhas que especifiquem o caminho de volta da escola até a casa de Luna.

Exemplos

Entrada	Saída
2 DIREITA QUINZE DIREITA QUATRO DIREITA ESCOLA ESQUERDA PRINCIPAL DIREITA ESCOLA	Vire a ESQUERDA na rua QUATRO. Vire a ESQUERDA na rua QUINZE. Vire a ESQUERDA na sua CASA. Vire a ESQUERDA na rua PRINCIPAL. Vire a DIREITA na sua CASA.

Observação: Cada uma das instruções deve, obrigatoriamente, seguir ao padrão anteriormente exemplificado. Note que, apesar de necessária na Língua Portuguesa, a acentuação foi eliminada das frases. O correto seria, por exemplo, “Vire à ESQUERDA na rua QUATRO.”.

Perceba também que todas as frases terminam com “.” (ponto final).

A saída de seu programa deve, portanto, ser *exatamente* igual à apresentada em cada caso de teste.

Entrada	Saída
3 ESQUERDA DEZ ESQUERDA OITO DIREITA CINCO DIREITA ESCOLA ESQUERDA DEZ ESQUERDA OITO ESQUERDA QUATRO DIREITA DOIS DIREITA ESCOLA DIREITA DEZ DIREITA OITO ESQUERDA QUATRO ESQUERDA ESCOLA	Vire a ESQUERDA na rua CINCO. Vire a ESQUERDA na rua OITO. Vire a DIREITA na rua DEZ. Vire a DIREITA na sua CASA. Vire a ESQUERDA na rua DOIS. Vire a ESQUERDA na rua QUATRO. Vire a DIREITA na rua OITO. Vire a DIREITA na rua DEZ. Vire a DIREITA na sua CASA. Vire a DIREITA na rua QUATRO. Vire a DIREITA na rua OITO. Vire a ESQUERDA na rua DEZ. Vire a ESQUERDA na sua CASA.

Entrada	Saída
1 ESQUERDA R6 ESQUERDA R4 ESQUERDA R9 DIREITA R7 ESQUERDA ESCOLA	Vire a DIREITA na rua R7. Vire a ESQUERDA na rua R9. Vire a DIREITA na rua R4. Vire a DIREITA na rua R6. Vire a DIREITA na sua CASA.

Entrada	Saída
1 DIREITA R1 DIREITA R2 DIREITA R4 DIREITA R3 ESQUERDA ESCOLA	Vire a DIREITA na rua R3. Vire a ESQUERDA na rua R4. Vire a ESQUERDA na rua R2. Vire a ESQUERDA na rua R1. Vire a ESQUERDA na sua CASA.



10 Notação polonesa inversa



(++)

Usualmente uma expressão aritmética e é escrita com os operadores binários entre os respectivos operandos e, por isso, essa notação é chamada de *infixa*.

Por exemplo:

$$e = (2 + 4 \cdot (6 - 1) / 2)$$

ou, equivalentemente:

$$e = \left(2 + 4 \cdot \frac{(6 - 1)}{2} \right)$$

Uma forma alternativa de escrever e é utilizando-se da *Notação Polonesa Inversa* (ou *Notação Posfixa*). Essa notação foi inventada pelo filósofo e cientista da computação australiano Charles Leonard Hamblin (1922-1985) em meados dos anos 1950, com base na *Notação Polonesa*, introduzida em 1920 pelo matemático polonês Jan Lukasiewicz (1878-1956).

A tabela a seguir mostra alguns exemplos de operações e suas notações:

EXPRESSÃO	NOTAÇÃO INFIXA	NOTAÇÃO POL. INVERSA
$a + b$	$a + b$	$ab+$
$a - b$	$a - b$	$ab-$
$a \cdot b$	$a \cdot b$	$ab*$
a^b	a^b	ab^{\wedge}
$\frac{a}{b}$	a/b	$ab/$
$\frac{a+b}{c}$	$(a + b) / c$	$ab + c /$
$\frac{a \cdot b - c \cdot d}{e \cdot f}$	$((a * b) - (c * d)) / (e * f)$	$ab * cd * - ef /$

Note que a ordem dos operandos é a mesma nas notações infix e posfixa. Além disso, a notação posfixa dispensa o uso de parênteses.

Tarefa

A sua tarefa é construir um programa de computador para *traduzir* expressões escritas com a notação infix para a notação posfixa.

As expressões são formadas por constantes numéricas, variáveis e operadores aritméticos. Para simplificar, considere que na expressão infixa são usadas apenas as operações aritméticas de soma, subtração, divisão, multiplicação e exponenciação (+, −, /, * e ^).

Considere também que nomes de variáveis são formados por apenas uma única letra maiúscula (A, B, C, ..., Z), as constantes numéricas são formadas por apenas um dígito (1, 2, ..., 9) e há um único espaço em branco, antes e após, um operador aritmético.

Se para cada símbolo de “abre parêntese” em uma expressão na notação infixa há um correspondente símbolo de “fecha parêntese”, diremos que a expressão está *bem-formada*. Do contrário ela é considerada *mal-formada*.

Entrada

Os dados de entrada são compostos por vários casos de teste, sendo que a primeira linha contém um único número natural t que indica o número de casos de teste, com $1 \leq t \leq 10$.

Cada caso de teste está numa linha da entrada e contém uma expressão aritmética escrita na notação infixa, com n operadores, sabe-se que $0 \leq n \leq 100$. Cada símbolo (numeral, variável, abre parêntese, fecha parêntese ou operador) é sempre precedido e sucedido por um único espaço em branco, exceto quando ele é o primeiro ou o último da expressão.

Saída

A saída, de cada caso de teste fornecido, deve ser apresentada numa linha e consiste da correspondente expressão aritmética escrita na notação posfixa ou, alternativamente, a mensagem “mal-formada”, caso a expressão na notação infixa seja *mal-formada*.

Da mesma forma que as entradas, cada linha da saída possui cada um dos seus símbolos (numeral, variável ou operador) precedido e sucedido por um único espaço em branco, exceto quando ele é o primeiro ou o último da expressão.

Exemplos

ENTRADA	SAÍDA
4	0 4 /
0 / 4	E 5 / 8 U ^+ 4 + 7 9 ^2 * +
(E / 5 + (8) ^U + 4) + 7 ^ (9) * 2	mal-formada
5 * 4 / (M + 8 + T − B * 8 ^T ^T	mal-formada
Q * H * S / 2 * (Q / (J) ^Y * C	

ENTRADA	SAÍDA
4	6 4 + 2 *
(6 + 4) * 2	6 2 ^2 2 ^- 7 *
(6 ^2 - 2 ^2) * 7	4 2 + 5 4 - * 2 ^
((4 + 2) * (5 - 4)) ^2	7 6 * 5 4 * +
7 * 6 + 5 * 4	

Atenção: Nas tabelas apresentadas anteriormente pode não parecer que há um ÚNICO ESPAÇO EM BRANCO entre cada um dos símbolos impressos, mas considere que HÁ.



11 Estação de trem



(+++)

Há uma famosa estação de trem na cidade *PopPush*. Esta cidade fica em um país de relevo incrivelmente acidentado e a sua estação foi criada no último século. Infelizmente os fundos financeiros eram extremamente limitados naquela época e, por isso, foi possível construir somente uma pista. Além disso, devido a problemas de espaço, foi feita uma pista apenas até a estação.

A tradição local é que todos os “comboios” que chegam vindo da direção *A* continuam na direção *B* com os vagões reorganizados, de alguma forma.

Suponha que o trem que está chegando da direção *A* tem n vagões numerados sempre em ordem crescente: $1, 2, 3, \dots, n$. O primeiro vagão que chega é o de número 1 e o último que chega é o de número n . Existe um “Chefe de Reorganização de Trens” que quer saber se é possível reorganizar os vagões para que os mesmos saiam na direção *B* na ordem $a_1, a_2, a_3, \dots, a_n$.

O *chefe* pode utilizar qualquer estratégia para obter a saída desejada. Por exemplo: se o chefe quer saber se a saída $5, 4, 3, 2, 1$ é possível em *B*, nesse caso, basta o *chefe* deixar todos os vagões entrarem na estação (do 1 ao 5) e, depois, retirar um a um: retira o 5, retira o 4, retira o 3, retira o 2 e, por último, retira o 1. Desta forma a resposta seria **sim**. O vagão que entra na estação só pode sair para a direção *B* e é possível incluir quantos vagões forem necessários para retirar o primeiro vagão desejado.

Entrada

A primeira linha da entrada contém $n \in \mathbb{N}^*$, o número vagões, com $1 \leq n \leq 1000$. Em cada uma das linhas seguintes – podem ser m linhas, com $1 \leq m \leq 10^6$, há uma permutação dos valores $1, 2, 3, \dots, n$ que representa a ordem de saída desejada pelo *chefe* da estação.

Saída

Para cada caso de teste fornecido, o programa deve imprimir, numa única linha, a mensagem **sim** se for possível obter a ordem de saída desejada, ou **nao**, do contrário.

Entrada	Saída
5 5 4 3 2 1	sim

Entrada	Saída
5 1 2 3 4 5	sim

Exemplos

Observação: Note que a palavra *não* está escrita em letras minúsculas e sem acentuação gráfica na saída de teste. É assim que você deverá imprimir.

Entrada	Saída
5 4 5 1 2 3	nao

Entrada	Saída
7 7 6 5 4 3 1 2	nao

Entrada	Saída
7 1 2 3 4 5 7 6	sim

Check for balanced parentheses in an expression

```
String s = "[()]{ }{ [() () ] () }";
```

```
= VALID STRING
```

```
String s = "[()]{ }{ [() () ] () }";
```

```
= INVALID STRING
```

12 Expressões



(++++)

Pedrinho e Zezinho precisam estudar a respeito da resolução de expressões matemáticas para uma prova que irão, em breve, prestar. Para isso, eles querem resolver muitos exercícios antes desta prova. Como sabem programar, então decidiram elaborar um “gerador de expressões matemáticas” – GEM – em \mathbb{C} . O gerador de expressões que eles criaram funciona em duas fases:

1ª fase – É gerada uma cadeia de caracteres c_i que contém apenas os caracteres $\{, [, (, \},]$ e $)$. Ou seja: abre chave, abre colchete, abre parêntese, fecha chave, fecha colchete e, por fim, fecha parêntese.

2ª fase – O gerador adiciona números e operadores na estrutura criada na primeira fase, tendo por objetivo formar expressões aritméticas.

Ao terminar sua execução, o GEM produz uma cadeia de caracteres c , que pode ser dita “bem definida” (ou válida) se atender às seguintes propriedades:

1. $c = \lambda$, ou seja, c é uma cadeia de caracteres vazia (não contém nenhum caractere).
2. c é formada por uma cadeia “bem definida” envolvida por pares “abre” e “fecha” de parênteses, colchetes ou chaves.
Portanto, se a cadeia S é “bem definida”, então as cadeias (S) , $[S]$ e S também são bem definidas.
3. c é formada pela *concatenação* de duas cadeias “bem definidas”.
Logo, se as cadeias x e y são “bem definidas”, a cadeia xy é “bem definida”.

Depois que Pedrinho e Zezinho geraram algumas expressões matemáticas, eles perceberam que havia algum erro na primeira fase do GEM, pois algumas das cadeias geradas **não eram** do tipo “bem definida”. Eles querem começar a resolver as expressões o mais rápido possível, e sabendo que você é um ótimo programador, resolveram pedir sua ajuda: escreva um programa que “*dadas várias cadeias geradas na primeira fase, determine quais delas são bem definidas e quais não são*”.

Entrada

A entrada é composta por diversas cadeias e, por isso, a primeira linha da entrada contém um inteiro t indicando o número de cadeias – $1 \leq t \leq 20$. Em seguida tem-se t linhas, cada uma com uma cadeia c . Sabe-se que o tamanho de c pode variar de 1 (um) a 1000 (mil) caracteres.

Saída

Para cada instância da entrada, imprima uma linha contendo a palavra “bem-formada” se a cadeia é deste tipo ou “mal-formada” caso contrário.

Exemplos

Entrada	Saída
12	bem-formada
()	bem-formada
([])	bem-formada
{ }	mal-formada
(]	mal-formada
} {	bem-formada
([{ }])	bem-formada
{ } [] ()	mal-formada
())	mal-formada
{ []	mal-formada
(bem-formada
(({ } { }) ([]) () { }) { }	mal-formada
((((((((({ ([])]))))))))	

Entrada	Saída
3	mal-formada
(([])	mal-formada
][][[bem-formada
(())	

Observação: Perceba todas as letras das palavras *bem-formada* e *mal-formada* estão grafadas em minúscula. Você deve fazer com que seu programa grafie *exatamente* desta maneira.