CADERNO DE PROBLEMAS

PROBLEMA A: NAPEBOOK

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

Cor: laranja

Limite de tempo: 1s

Descrição do problema

Empolgado com a possibilidade de se tornar uma celebridade da área de tecnologia, mas sem muita criatividade, você resolveu criar uma ferramenta revolucionária, mas não muito: uma rede social. Mais uma vez sem ideias totalmente originais, você a chamou de Napebook.

Após dois anos do lançamento, sua rede atingiu a marca espetacular de 100 usuários. Apesar de sua animação com o recorde, algo estranho começou a acontecer: aparentemente você atraiu algum maníaco homicida para sua pequena comunidade...

Um assassino serial, de codinome RS232, aparentemente, está usando a lista de nomes dos usuários do seu site e matando, na ordem da listagem, os usuários! O terrível malfeitor já chegou na letra O... Você está muito preocupado, sendo o seu nome Paul Grama...

Sabendo que a lista está em ordem alfabética, você resolve, de maneira muito discreta, alterar a ordenação alfabética para que o seu nome fique no final da lista. Esta mudança seria sutil o suficiente para não entregar a sua manobra não muito altruísta. Assim, a polícia teria tempo que pegar o infeliz meliante! Você resolveu utilizar um “novo” alfabeto, disposto na seguinte ordem:

q , a , z , w , s , x , e , d , c , r , f , v , t , g , b , y , h , n , u , j , m , i , k , o , l , p

Você está criando um programa para testar a nova ordenação. É preciso ter certeza de que funciona antes de publicar a alteração no Napenook, afinal, um erro pode ser fatal!

Entrada

A entrada é constituída de vários casos de teste. Cada caso de testes possui uma frase de no máximo 600 caracteres. As palavras de uma frase são separadas por espaços em branco simples. Cada palavra contém apenas letras minúsculas entre 'a' e 'z'. A entrada termina em EOF.

Saída

Para caso de teste, imprima uma linha contendo o catálogo ordenado.

Exemplos de testes

Entrada	**Saída**
wsnmx fvjxrdfd nqxhxs mlzlpk glrxnsh fwkp cxlzh pbzkzz vgpkhp gwpw kwd xwmdpd vhvkh pvdpfpsl qfrj bjlnrn dsr mdj nlwn ptjvn	wsnmx fvjxrdfd glrxnsh nqxhxs mlzlpk cxlzh fwkp vgpkhp gwpw pbzkzz qfrj xwmdpd vhvkh kwd pvdpfpsl dsr bjlnrn nlwn mdj ptjvn

PROBLEMA B: OPERAÇÃO ILEGAL

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

Cor: branco

Limite de tempo: 1s

Descrição do problema

No aclamado filme de Charlie Chaplin "O Garoto", uma cena ao mesmo tempo cômica e crítica é a das janelas. Nessa cena, o garoto quebra janelas atirando pedras e a personagem de Charlie Chaplin oferece o serviço de reparo. Adubervaldo assistiu o filme e decidiu, distorcendo a crítica de Charlie Chaplin, expandir seus negócios. Ele é um borracheiro e resolveu contratar algumas crianças para furarem os pneus dos carros estacionados no shopping center do qual é vizinho. Antes de começar a operação, ele resolveu primeiro avaliar se o risco vale a pena.

Para avaliar a lucratividade do negócio ele pediu que as crianças contassem as rodas dos carros e motos do estacionamento no horário de pico. Após o retorno das crianças, Adubervaldo lembrou que pneus de carro são mais lucrativos do que os de moto então agora ele precisa avaliar todas as possíveis situações de quantidades de carros e motos. Se as crianças contarem 6 rodas então Adubervaldo teria que avaliar a lucratividade da situação de 1 carro e 1 moto estarem estacionados assim como a situação de 3 motos estarem estacionadas (considerando que cada carro possui 4 rodas e cada moto possui 2 rodas). Antes de avaliar cada possível situação ele precisa saber primeiro quantas existem. Você, sem conhecer essa operação, foi contratado para criar um programa que calcula, dada a quantidade de rodas, o número de situações possíveis.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por uma linha contendo um inteiro Q ($1 \leq Q \leq 2000000000$), a quantidade de rodas que as crianças contaram no estacionamento. A entrada termina quando $Q = 0$.

Saída

Para cada caso de teste, exiba uma linha contendo um inteiro X que representa o número de situações possíveis.

Exemplos de testes

Entrada	**Saída**
6 22 44 2 0	2 6 12 1

PROBLEMA C: CAKE PARTY

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

Cor: vermelho

Limite de tempo: 1s

Descrição do problema

Seu Juca é um pai de família dedicado, e também um homem que possui muitas filhas. Uma coisa em comum entre todas elas é a paixão por um suculento pedaço de bolo.

Existem, porém, vários tipos de bolos; cada filha gosta mais de um tipo do que de outro. Por exemplo, para saber se uma filha gosta mais de um bolo do que de outro, ela geralmente atribui uma nota para ele, um inteiro N com valor de 0 a 5, sendo que 0 significa que ela não gosta do bolo, e 5 significa que ela gosta bastante do bolo.

Para satisfazer uma determinada filha, é preciso que Seu Juca compre bolos cujas notas somadas sejam iguais ou superiores a 5.

Seu Juca é um morador da cidade de Xundanópolis e, infelizmente, os bolos costumam ser um pouco caros por lá. Sendo assim, Seu Juca gostaria de gastar o menos possível para comprar os bolos, e ainda assim conseguir satisfazer todas as suas filhas.

Vale lembrar que cada bolo é bem grande e sempre consegue servir todas as filhas de Seu Juca, para a alegria dessa família amante de bolos gostosos!

Para entender melhor o problema, veja a tabela abaixo, que contém cada bolo disponível, seu preço P e sua nota N, de acordo com cada filha:

# Filha	# Bolo	Bolo 1 – R$9,50	Bolo 2 – R$5,00	Bolo 3 – R$4,00
1	**Notas:**	5	3	2
2		5	2	3

No exemplo acima, Seu Juca poderia comprar somente o bolo 1 (gastando R$9,50), que agradaria ambas as filhas. Porém, sairia um pouco mais barato se ele comprasse os bolos 2 e 3, que também agradariam ambas as filhas, e o gasto total seria R$9,00.

Sua tarefa é escrever um programa que ajude Seu Juca a decidir quais bolos ele deve comprar, de forma que gaste o menos possível e agrade todas as suas filhas.

Entrada

A primeira linha da entrada consiste de um inteiro T, indicando o número de casos de teste. Para cada caso de teste, a primeira linha contém dois inteiros F e B, indicando, respectivamente, o número total de filhas e o número de bolos existentes. A segunda linha contém B números decimais indicando os preços dos bolos existentes. A seguir, F linhas serão recebidas, cada uma contendo B números inteiros, indicando a nota da F-ésima filha, para o B-ésimo bolo.

Para os casos de teste da entrada, sempre haverá somente uma única forma de comprar os bolos com o menor gasto possível.

Saída

A saída deverá conter apenas uma linha no formato “Teste #A: B1 B2 ... Bn”, onde A indica o número do caso de teste, e os inteiros seguintes B1, B2, ..., Bn indicam quais bolos deverão ser comprados.

Limites

$1 \leq T \leq 25$

$1 \leq F \leq 10$

$1 \leq B \leq 20$

$0 \leq N \leq 5$

$1{,}0 \leq P \leq 10{,}0$

Exemplos de testes

Entrada	**Saída**
5 2 3 8.5 5 4 5 4 2 5 2 3 2 3 9.5 5 4 5 3 2 5 2 3 1 2 10 20 5 5 1 3 9.5 5 4 5 3 2 4 5 10 9 5 4 2 5 0 3 0 0 5 4 0 3 0 4 3 0 0 2 0 2 3 2 3	Teste #1: 1 Teste #2: 2 3 Teste #3: 1 Teste #4: 2 3 Teste #5: 1 4 5

PROBLEMA D: EXISTE VIDA APÓS O GT?

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

Cor: amarelo

Limite de tempo: 1s

Descrição do problema

Todos adoram Dragon Ball Z. Há algumas opiniões diversas quanto a versão original (sem o Z). Agora, o que todos concordam, é que o Dragon Ball GT é muito ruim. Depois desse desastre que manchou sua franquia, Akira Toriyama resolve voltar aos roteiros para recuperar a moral da série, lançando o revolucionário Dragon Ball Generation Y.

Em Generation Y, o tataraneto de Goku, Emu, nascido no planeta terra na década de 80, membro da famosa geração Y, resolve procurar as esferas do dragão para que possa pedir uma máquina que faz tudo por ele, afinal, ele não está muito disposto a estudar ou trabalhar.

Infelizmente Shenlong, o dragão que concede o desejo, também mudou com o tempo, e começou a fazer piadinhas e pegadinhas. Seu grande sonho é um dia trabalhar com stand up. Para “trollar” Emu, ele decide distribuir várias esferas repetidas pelo planeta, complicando o funcionamento do lendário GPS localizador de esferas.

Emu não gosta muito de pensar, então resolveu chamar você, seu amigo programador, para ajuda-lo. Sua missão é auxiliá-lo na tarefa de encontrar quantas esferas falsas existem. Vale lembrar que ao ter mais de uma esfera de um mesmo número, uma é falsa.

Entrada

A entrada será composta de vários casos de teste. Cada caso de teste é composto por uma linha com vários valores entre 1 e 7 que representam cada uma das esferas encontradas. A quantidade máxima de esferas encontradas é 100. A entrada termina em EOF.

Saída

Seu programa deve imprimir uma linha para caso de teste contendo o número da esfera juntamente com a quantidade de versões **falsas** que existem para aquela esfera conforme o exemplo abaixo.

Exemplos de testes

Entrada	**Saída**
1 7 2 6 5 3 4 3 1 5 2 6 4 3 2 5 4 6 7 1 2 3 5 6 3 3 3 3 2 4 5 6 1 2 7	1->1 | 2->1 | 3->2 | 4->1 | 5->1 | 6->1 2->1 | 3->1 | 5->1 | 6->1 2->1 | 3->2

PROBLEMA E: OLD MACDONALD HAD A FARM

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

Cor: azul

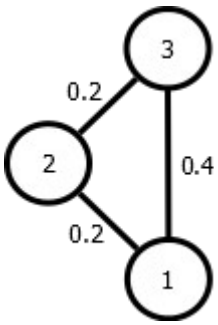
Limite de tempo: 1s

Descrição do problema

"Old MacDonald Had a Farm" é uma canção infantil sobre um fazendeiro chamado MacDonald que tinha muitos animais em sua fazenda. Algo que poucos sabem é que essa canção foi baseada em uma pessoa. Essa canção é antiga e, infelizmente, o senhor MacDonald não está mais vivo. Sua fazenda foi passada de geração em geração e, atualmente, seu bisneto tenta mantê-la funcionando. Ele está tendo dificuldades financeiras e quer diminuir seus gastos. Um grande problema que ele tem é o transporte dos ovos produzidos até as cidades onde os vende. Ao transportar os ovos, um certo percentual é perdido ou quebrado devido às condições precárias das estradas por onde passa.

Ele quer minimizar essas perdas para poder economizar e continuar a trabalhar na fazenda de seu bisavô. Ele anotou o percentual de ovos que perde ao passar por cada estrada mas agora está com dificuldades para escolher qual caminho deve tomar para diminuir suas perdas.

Você, comovido com a situação, resolveu ajudá-lo. Você decidiu criar um programa que, dada a quantidade de cidades pelas quais ele pode passar e a lista de estradas e suas respectivas perdas percentuais, calcula a menor perda percentual possível. Para facilitar o problema você decidiu enumerar as cidades de 1 a N e considerar que todas as estradas são de mão dupla. Também tomou o cuidado para que a cidade de origem fosse sempre 1 e a destino fosse N. Um exemplo do problema é ilustrado na figura abaixo.



Nela, o bisneto de MacDonald quer ir da cidade 1 para a cidade 3; logo ele pode escolher um dos seguintes caminhos:

- Ir até a cidade 2 e então ir para 3;
- Ir diretamente para a cidade 3.

No primeiro caso a sua perda percentual seria 36.00% já que no trecho entre as cidades 1 e 2 ele perderia 20% do total e no segundo trecho perderia 20% do restante. No segundo caso a perda seria de 40.00%. Então, mesmo sendo um caminho maior, é preferível passar antes pela cidade 2.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por várias linhas. A primeira linha de um caso de teste contém dois valores inteiros N e M que representam a quantidade de cidades e a quantidade de estradas, respectivamente. As M

linhas seguintes contém dois valores inteiros A e B que representam as cidades que a estrada liga e um valor decimal P que representa o percentual de perda ao transitar por aquela estrada. A entrada termina quando N = M = 0.

Saída

Para cada caso de teste, exiba uma linha contendo o menor percentual de perda possível com arredondamento na segunda casa decimal.

Limites

$1 \leq N \leq 1000$

$1 \leq M \leq N*(N-1)/2$

$1 \leq A, B \leq N$

$0,00 \leq P \leq 1,00$

Exemplos de testes

Entrada	**Saída**
3 3 1 2 0.20 2 3 0.20 1 3 0.40 0 0	36.00%

PROBLEMA F: MÉTODO BRAILLE

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

Cor: preto

Limite de tempo: 1s

Descrição do problema

O Sistema Braille, utilizado universalmente na leitura e na escrita por pessoas cegas, foi inventado na França por Louis Braille, um jovem cego, reconhecendo-se o ano de 1825 como o marco dessa importante conquista para a educação e a integração dos deficientes visuais na sociedade. Antes desse histórico invento, registram-se inúmeras tentativas em diferentes países, no sentido de se encontrarem meios que proporcionassem às pessoas cegas condições de ler e escrever. Dentre essas tentativas, destaca-se o processo de representação dos caracteres comuns com linhas em alto relevo, adaptado pelo francês Valentin Hauy, fundador da primeira escola para cegos no mundo, em 1784, na cidade de Paris, denominada Instituto Real dos Jovens Cegos.

Louis Braille, ainda jovem estudante, tomou conhecimento de uma invenção denominada sonografia ou código militar, desenvolvida por Charles Barbier, oficial do exército francês. O invento tinha como objetivo possibilitar a comunicação noturna entre oficiais nas campanhas de guerra. Baseava-se em doze sinais, compreendendo linhas e pontos salientes, representando sílabas na língua francesa. O invento de Barbier não logrou êxito no que se propunha, inicialmente. O bem intencionado oficial levou seu invento para ser experimentado entre as pessoas cegas do Instituto Real dos Jovens Cegos. A significação tátil dos pontos em relevo do invento de Barbier foi a base para a criação do Sistema Braille.

Apesar de algumas resistências mais ou menos prolongadas em países da Europa e nos Estados Unidos, o Sistema Braille, por sua eficiência e vasta aplicabilidade, se impôs definitivamente como o melhor meio de leitura e de escrita para as pessoas cegas.

O método consiste em um alfabeto de pontos em relevo, que são organizados em uma tabela com três linhas e duas colunas formando um retângulo, onde pelo menos um se destaca em relação aos demais. As combinações desses pontos dispostos estão relacionados a símbolos que representam letras simples e acentuadas, pontuações, símbolos, notas musicais, sinais algébricos entre outros, propiciando ao deficiente visual a leitura e escrita de qualquer texto.

O método admite um número finito de caracteres, pois os pontos em relevo são posicionados em diferentes lugares, ou seja, para se calcular a quantidade de caracteres que é possível representar, fazemos a combinação de seis pontos: um a um, dois a dois, três a três, quatro a quatro, cinco a cinco e seis a seis. E então somamos essas combinações, por exemplo, para seis pontos temos como resultado 63 caracteres.

Digamos que você possa inventar sistemas com uma quantidade diferente de pontos, sua equipe foi contratada para calcular a quantidade de caracteres que é possível representar dado uma quantidade x de pontos.

Entrada

A entrada é constituída de vários casos de teste. Cada caso de teste possui um número inteiro entre 1 e 60 que representa a quantidade de pontos. O programa deverá ser encerrado quando informado uma quantidade de pontos inexistente.

Saída

A saída será composta por um número inteiro que representa a quantidade de caracteres que é possível representar.

Exemplos de testes

Entrada	Saída
5 6 3 8 0	31 63 7 255

PROBLEMA G: IMPLEMENTAÇÃO DO RAID 5

Arquivo: `raid5.[cpp/c/java]`

Cor: cinza

Limite de tempo: 1s

Descrição do problema

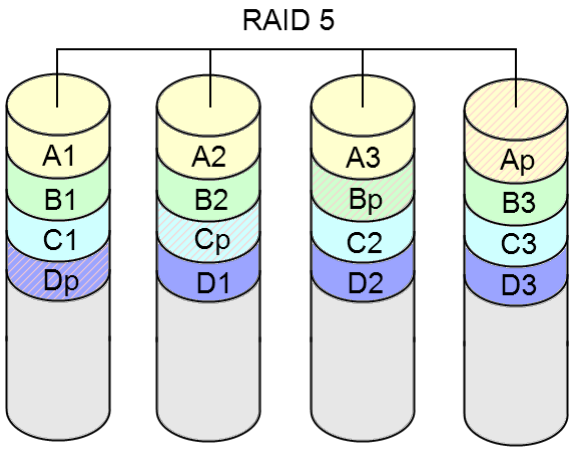
RAID (Redundant Array of Inexpensive Disks - Conjunto Redundante de Discos Econômicos) é um meio de se criar um sub-sistema de armazenamento composto por vários discos individuais com a finalidade de ganhar segurança e desempenho. RAID seriam compostos por dois ou mais discos trabalhando simultaneamente para um mesmo fim.

Há várias implementações possíveis de RAID dividida em alguns modos:

- **RAID 0** - É uma simples concatenação de partições para criar uma grande partição virtual.
- **RAID 1** - é o nível de RAID que implementa o espelhamento de disco, também conhecido como mirror.
- **RAID 2** – trabalha com controle de erro ECC
- **RAID 3** – é uma versão simplificada do RAID nível 2.
- **RAID 4** – funciona com dois ou mais discos iguais. Um dos discos guarda a paridade (uma forma de soma de segurança) da informação contida nos discos.
- **RAID 5** – funciona similarmente ao RAID 4, mas supera alguns dos problemas mais comuns sofridos por esse tipo. As informações sobre paridade para os dados do conjunto são distribuídas ao longo de todos os discos do conjunto, ao invés de serem armazenadas num disco dedicado, oferecendo assim mais desempenho que o RAID 4, e, simultaneamente, tolerância a falhas.

Uma empresa precisa elaborar o software que vai operar em uma controladora de discos com suporte a RAID 5. Para isso, a princípio você deve elaborar um algoritmo simplificado que será utilizado em simulações e testes.

O funcionamento do RAID 5 é o seguinte. Para cada faixa de dados, um disco é usado para armazenar a paridade calculada para aquela faixa, como mostra a figura.



Na primeira faixa de dados (A), o quarto disco é usado para armazenar a paridade .

Na segunda faixa (B), o terceiro disco é usado para paridade, e assim sucessivamente.

Para a simulação, cada bloco de dados será composta de 4 bits, e o conjunto será composto de 5 discos.

Para calcular o bloco de paridade, segue o exemplo:

Dados a serem gravados em hexadecimal: A2B7 (cada algarismo corresponde a um bloco de um disco).

```
Bloco 1: A(16) = 1010(2)
Bloco 2: 2(16) = 0010(2)

Bloco 3: B(16) = 1011(2)

Bloco 4: 7(16) = 0111(2)
```

Para cada conjunto de bits, a paridade é obtida da seguinte forma. Somam-se os bits. Se o resultado é PAR, a paridade é 0. Se o resultado é ÍMPAR, a paridade é 1.

```
1010
0010
1011
0111
----
0100 ← Bloco de paridade
```

para a faixa específica.

$0100_{(2)} = 4_{(16)}$

Portanto, para o conjunto de blocos A2B7, o bloco de paridade é 4.

$\text{Bloco p: } 4_{(16)} = 0100_{(2)}$

No simulador, serão considerados 5 discos e 16 faixas de gravação. A tabela a seguir demostra a posição dos blocos de paridade (P) para cada faixa:

Discos	**0**	**1**	**2**	**3**	**4**
Faixa 0					P
Faixa 1				P	
Faixa 2			P		
Faixa 3		P			
Faixa 4	P				
Faixa 5					P
Faixa 6				P	
Faixa 7			P		
Faixa 8		P			
Faixa 9	P				
Faixa 10					P
Faixa 11				P	
Faixa 12			P		
Faixa 13		P			
Faixa 14	P				
Faixa 15					P

Tabela1: Posição dos Blocos de Paridade

Esse simulador aceitara vários comandos de entrada, tais como escrita de dados (W), leitura de dados (R), limpeza de bloco (C), saída (Q) e importação (I) que simula a existência de dados nos discos. Inicialmente, considere que todas as faixas estão livres, isto é, todos os discos estão vazios (preenchidos com 0). Teremos as operações:

Operação de importação (I): recebe o valor dos 5 blocos e grava na próxima faixa livre, marcando a mesma como ocupada, sem se preocupar com paridade. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: IA2B27 Saída: 01A2B27`

Operação de gravação (W): recebe o valor dos 4 blocos de dados, calcula a paridade e grava os dados na próxima faixa livre, marcando a mesma como ocupada, nos discos corretos, de acordo com a posição da faixa. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: WA2B7 Saída: 02A24B7`

Operação de leitura (R): recebe o número da faixa a ser lida, faz a leitura e verifica se a paridade está correta. Retorna os blocos lidos. Caso a paridade esteja errada, apresenta a palavra 'FALHA' na frente dos dados.

Ex: `Entrada: R02 Saída: A2B7`

Operação de limpeza (C): recebe o número da faixa a ser limpa, zera todos os discos e marca a faixa como livre.

Ex: `Entrada: C01 Saída: 0100000`

Operação de saída (Q): encerra a execução do programa.

Ex: `Entrada: Q Saída:`

Entrada

A entrada é sempre composta de uma letra maiúscula que representa a operação, seguida do parâmetro esperado por essa operação. Para a operação de importação, o parâmetro é composto de 5 dígitos hexadecimais. Para a operação de escrita, o parâmetro é composto de 4 dígitos hexadecimais. Para as operações de leitura e limpeza, o parâmetro é composto de 2 dígitos decimais. Para a operação de saída, não há parâmetros.

Saída

Para as operações de importação (I) e escrita (W), a saída é composta de 2 dígitos decimais (faixa) e 5 hexadecimais (blocos) ou 'CHEIO' no caso do disco estar cheio. Para a operação de leitura (R) a saída é composta de 4 dígitos hexadecimais e, no caso de erro de paridade, seguido da palavra 'FALHA'. Para a operação de limpeza (C), a saída é composta de 2 dígitos decimais (faixa) e 5 dígitos zeros. Para a operação de saída (Q), não há saídas.

Exemplos de testes

Entrada	Saída
I12344	0012344
I63207	0163207
IFA3CA	02FA3CA
IA3AAA	03A3AAA
I0BBBB	040BBBB
ICCCC0	05CCCC0
IDDD0D	06DDD0D
IEE2EE	07EE2EE
IF0FFF	08F0FFF
I0ABCD	090ABCD
I8777F	108777F
I98765	1198765
IAB6BC	12AB6BC
IC0DBA	13C0DBA
I11101	1411101
I20222	1520222
I00000	CHEIO
R00	1234
R03	AAAA-FALHA
R05	CCCC
R07	EEEE-FALHA
R09	ABCD
R11	9875-FALHA
C05	0500000
C02	0200000
C07	0700000
R05	0000
R07	0000
WCAC0	02CAAC0
W4359	054359B
WF123	07F1F23
W0201	CHEIO
C10	1000000
W0201	1002013
W1312	CHEIO
R02	CAC0
R05	4359
R07	F123
Q	

PROBLEMA H: MATRIZ MALUCA

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

Cor: verde

Limite de tempo: 1s

Descrição do problema

Seu programa deve processar uma matriz de 1000 x 1000 que tem vários elementos iguais a “1” e reescrevê-la. Para cada célula igual a “1”, seu programa deve preencher a coluna inteira dessa célula com “1” e a linha inteira dessa célula também com “1”. No final, ele deve mostrar o novo número de “1”s na matriz.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste começa com uma linha contendo um valor N (1 ≤ N ≤ 10000) que representa a quantidade de "1"s na matriz original. Seguem N linhas cada uma com dois valores separados por espaço que representam a célula de um valor "1" na matriz original.

A entrada termina quando N = 0.

Saída

Para cada caso de teste seu programa deve imprimir uma linha contendo a quantidade de "1"s na matriz após o processamento.

Exemplos de testes

Entrada	**Saída**
3 1 1 1 4 3 3 5 3 3 3 5 5 5 5 8 10 10 0	4994 6988

PROBLEMA I: FIFA ULTIMATE TEAM

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

Cor: roxo

Limite de tempo: 1s

Descrição do problema

Brasil é o país de futebol, ano de copa chegando, e para aqueles que não tem o talento da bola mas gostam do esporte, só nos resta jogar Fifa.

Afinal, antes um futebol virtual na mão do que um coração infartando!

O melhor de tudo é que nas gerações atuais dos consoles, não existe mais aquele problema de jogar apenas contra a CPU, podemos jogar online!!!

Existem diversos modos, para todos os gostos, mas um que gera fortes emoções é jogar o Ultimate Team, no qual é possível montar seu time com jogadores de diversos clubes e jogar online.

Uma das caractéristicas destes times é que eles precisam manter um certo nível de entrosamento, para que o time tenha um bom rendimento.

As características que são avaliadas, de forma simplificada são as seguintes:

- A relação entre o jogador da posição e jogadores da mesma liga
- A relação entre o jogador da posição e jogadores do mesmo clube
- A relação entre o jogador da posição e jogadores da mesma nacionalidade

O cálculo do entrosamento se através da soma das seguintes características:

- Total de jogadores da mesma liga relacionados a posição
- Total de jogadores do mesmo clube relacionados a posição
- Total de jogadores do mesmo país relacionados a posição

Este valor deve ser ponderado de acordo com o número de relações de cada posição, evitando vantagem para uma posição com mais relacionamentos.

O critério de desempate é a ordem em que os jogadores aparecem.

Exemplo: A posição GOL do exemplo de entrada possui relação com as posições ZAG1 e ZAG2.

O jogador selecionado para a posição GOL terá seus atributos (liga, clube e nacionalidade) comparados aos jogadores selecionados para as posições ZAG1 e ZAG2.

A soma de itens em comum é o resultado utilizado, e este deve ser proporcionalizado pelo número de relações que a posição GOL possui (no caso 2).

É muito fácil montar times com jogadores de uma única liga / clube / nacionalidade, mas o complicado é criar times multi ligas e manter um bom nível de entrosamento.

É aí meu caro amigo que você vai me ajudar!

Você receberá a configuração do time, a relação entre as posições e os jogadores disponíveis para cada posição.

Quero receber de volta o melhor time possível dada esta formação e jogadores.

Entrada

A entrada é composta de uma variável contendo o número de casos de teste (0 <= N <= 100), seguidas por 11 posições.

Cada posição é composta por um identificador único com no máximo 5 caracteres, por um inteiro K (1 <= K <= 5) determinando o número de jogadores disponíveis para esta posição e uma string contendo o relacionamento com outras posições.

Após receber a posição, é necessário receber os K jogadores, sendo que para cada jogador serão informados em uma única linha:

- O nome do jogador com no máximo 5 letras;
- O país de origem com 3 letras;
- O clube no qual atua com 3 letras;
- A liga em que atua com 3 letras.

Saída

Para cada caso de teste devem ser exibidas 11 linhas contendo a posição na ordem em que foi informada apresentando o identificador da posição e o nome do jogador escolhido.

Exemplos de testes

Entrada	**Saída**
1 GOL 3 ZAG1 ZAG2 Neuer ALE BAY ALE Cech RCH CHE ING Valde ESP BAR ESP LATD 2 ZAG1 MC1 MC2 Lahm ALE BAY ALE Sergi ESP RMA ESP LATE 1 ZAG2 MC1 MC2 Marce BRA RMA ESP ZAG1 2 GOL LATD Pepe ESP BAR ESP RVar FRA RMA ESP ZAG2 1 GOL LATE Pique ESP BAR ESP MC1 3 LATD LATE MC2 MC3 March ITA JUV ITA Oscar BRA CHE ING Xabi ESP RMA ESP MC2 2 LATE LATD MC1 MC3 Pirlo ITA JUV ITA Inies ESP BAR ESP MC3 2 MC1 MC2 ATA1 ATA2 ATA3 Pogba ITA JUV ITA Xavi ESP BAR ESP ATA1 2 MC3 ATA1 ATA3 Neyma BRA BAR ESP Messi ARG BAR ESP ATA2 3 MC3 ATA1 ATA3 Tevez ARG JUV ITA DCost ESP ATM ESP Sanch ESP BAR ESP ATA3 2 MC3 ATA1 ATA2 ATA3 Ibra SUE PSG FRA Pedro ESP BAR ESP	GOL: Valde LATD: Serg LATE: Marce ZAG1: Pepe ZAG2: Pique MC1: Xabi MC2: Inies MC3: Xavi ATA1: Neyma ATA2: Sanch ATA3: Pedro