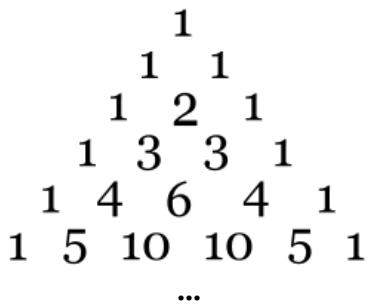
Problema A

Triângulo de Pascal

Arquivo fonte: `pascal.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

O Triângulo de Pascal é um triângulo numérico formado por coeficientes binomiais, conforme imagem seguinte.



Nele, o valor da primeira e última coluna de cada linha é igual a um (1). O valor das outras posições pode ser obtido a partir da soma do valor que está logo acima da posição de interesse com o que está à esquerda deste último. Como exemplo, convencionando que a primeira linha é L=0 e a primeira coluna de cada linha é C=0, o elemento (L=4, C=2) = (L=3, C=2) + (L=3, C=1) = 3+3 = 6.

Seu professor de algoritmos e lógica de programação pediu para você desenvolver um programa que, dados uma linha L e uma coluna C, retorna o valor do Triângulo de Pascal nesta posição.

Entrada

A entrada é composta por vários casos de teste. A primeira linha da entrada contém o número N, $1 \leq N \leq 5000$, de casos de teste. As N linhas seguintes contêm dois inteiros: L, $0 \leq L < 25$, e C, $0 \leq C \leq L$, separados por um espaço, especificando a posição do Triângulo de Pascal para a qual se deseja obter o valor.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro representando o valor do Triângulo de Pascal na posição especificada.

Exemplos

Exemplo de entrada

```
3
0 0
4 2
1 1
```

Saída para o exemplo de entrada

```
1
6
1
```

Solução

Trata-se de um problema que pode ser resolvido com recursão. No próprio enunciado, as regras de recursão, incluindo os critérios de parada, estão especificados. O método recursivo deve verificar inicialmente se os parâmetros de entrada correspondem à coluna=0 ou linha=coluna e, em caso

afirmativo, retornar 1. Caso contrário, o método deve retornar uma chamada recursiva da forma: metodoRecursivo(linha - 1, coluna) + metodoRecursivo(linha - 1, coluna - 1).

O problema também pode ser resolvido de maneira não recursiva e, nesse caso, seria necessário criar um *array* com as dimensões máximas previstas para o triângulo (25 linhas e 25 colunas) e computar os valores a partir da especificação constante no enunciado (primeira e última colunas úteis da linha com 1, as restantes com os valores calculados a partir dos dados da linha anterior). Após preenchido o *array*, bastava receber os valores da entrada e exibir o valor contido na posição correspondente do *array*. É uma solução que requer mais código do que a solução recursiva (demandando mais tempo antes de submeter a solucao), porém pode ser muito rápida em tempo de execução, sendo indicada em situações com *time limit* muito apertado.

Problema B

Quantas vagas?

Arquivo fonte: `vagas.{c |cpp,| java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Josevaldo é o responsável pelo setor de serviços gerais da Indústrias XPTO S/A. Como tem ocorrido na maioria das cidades brasileiras, o aumento do número de veículos em circulação cria constantemente problemas variados, entre eles o da falta de vagas para estacionamento na empresa.

O bolsão de estacionamento interno está sendo insuficiente para acomodar todos os veículos dos funcionários, então será liberado o estacionamento junto ao meio-fio do lado esquerdo de um quarteirão de uma alameda existente atrás do prédio dos escritórios administrativos. Josevaldo deve pintar os limites das vagas onde os carros podem estacionar.

Uma pergunta óbvia é: quantos carros poderão ser estacionados ali? Como é uma via de circulação (ainda que interna da empresa), deve-se deixar livre um trecho de 3 metros no início e no final do trecho, para não prejudicar a visão de quem vem no cruzamento. O espaço restante é livre para estacionamento, pois a via é comprida, reta, plana e larga. Sabendo-se que uma vaga ocupa 5 metros de comprimento, é fácil então calcular a quantidade de novas vagas disponível para estacionamento se soubermos o comprimento do quarteirão liberado para uso. Sua tarefa é, dado esse comprimento, determinar o número de carros que poderão estacionar ali.

Entrada

A entrada contém vários casos de teste a serem processados. Cada um dos casos de teste é composto por um inteiro C, $0 \leq C \leq 1000$, que indica o comprimento total do lado esquerdo do quarteirão liberado para estacionamento. O último caso de teste é seguido por uma linha com o número -1.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro indicando a quantidade de vagas de estacionamento disponível no novo local.

Exemplos

Exemplo de entrada

```
506
122
8
-1
```

Saída para o exemplo de entrada

```
100
23
0
```

Solução

O problema mais fácil da prova requeria apenas que, para o inteiro C lido, se descontasse os 6 metros de recuo e depois se dividisse o que sobrou por 5, que era o tamanho de cada vaga. Como C é um inteiro, 6 é um inteiro e 5 também é um inteiro, a divisão a ser realizada é também uma divisão de inteiros, produzindo o número de vagas sem casas decimais, como requerido pelo problema. Prestar atenção apenas à possibilidade de C ser menor que 6: como não existe quantidade negativa de vagas de estacionamento, imprimir zero como resposta.

Problema C

Nomes ordenados

Arquivo fonte: `ord.{c | cpp | java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

José tem uma relação com o primeiro nome de algumas pessoas. Ele gostaria de colocá-los em ordem, mas não da maneira convencional. O critério de ordenação que ele deseja coloca os nomes mais curtos antes dos nomes mais compridos. Os nomes de mesmo comprimento ficam em ordem alfabética, lembrando que maiúsculas têm precedência sobre as minúsculas. Sua tarefa é, para uma relação de nomes indicada por José, produzir a sequência dos nomes ordenados conforme o critério descrito anteriormente.

Entrada

A entrada é iniciada por um valor N, $0 \leq N \leq 50$, que indica a quantidade de casos de teste a serem processados. Cada um dos N casos de teste inicia-se com um inteiro P, $0 \leq P \leq 100$, que indica a quantidade de nomes de pessoas a serem ordenados. Seguem-se P linhas, cada uma com um nome de até 20 caracteres de comprimento, composto apenas por letras maiúsculas e minúsculas.

Saída

Para cada caso de teste, imprima uma linha contendo a relação de nomes ordenada pelo critério indicado por José, com uma vírgula e um espaço em branco separando os nomes, conforme os exemplos apresentados a seguir. Imprimir um ponto final e uma quebra de linha após cada caso de teste.

Exemplos

Exemplo de entrada

```
2
6
Roberto
Roberval
Ari
Rogerio
Ana
antonio
10
Claudio
Eva
Adao
Roberto
Roberval
Ari
Rogerio
Adelia
Oto
Zoraide
```

Saída para o exemplo de entrada

```
Ana, Ari, antonio, Roberto, Rogerio, Roberval.
Ari, Eva, Oto, Adao, Adelia, Claudio, Roberto, Rogerio, Zoraide, Roberval.
```

Solução

Este problema de ordenação é relativamente fácil, mas foi prejudicado pela impressão do resultado incorreto no primeiro caso de teste (o correto é `Ana, Ari, Roberto, Rogerio, antonio, Roberval`.), embora tivesse sido indicado nas respostas das *clarifications* da prova. O enunciado, por sua vez, está correto, e se fosse seguido à risca, levaria a um programa que produziria a saída esperada pelos juízes.

Qualquer método de ordenação poderia ser usado, já que o volume de dados de cada caso de teste era pequeno. Ao comparar dois elementos do vetor, verificar inicialmente o comprimento de cada um, se forem diferentes, colocar o mais curto antes do mais comprido, independentemente do conteúdo de cada *string*. Se, por outro lado, os elementos forem de comprimentos iguais, usar o resultado da comparação entre as st*r*ings, que por não ser *case-sensitive*, coloca automaticamente as maiúsculas antes das minúsculas, como solicitado no enunciado do problema.

Problema D

Cadeado

Nome do arquivo fonte: `cadeado.{c | cpp | java` *ou* `pas}`

Autor: Anderson Viçoso de Araújo (Fatec São José dos Campos)

O professor doutor Papadopoulos tem origem grega, mas nasceu no Brasil. Ainda jovem, descobriu a sua paixão pela música e pela vida acadêmica. Com isso, decidiu lecionar música. Após anos de estudo, passou em um concurso em uma universidade de média expressão do centro-oeste brasileiro. A universidade possui uma sala grande e agradável para o convívio diário entre os professores. Nesta sala, existem máquinas de café (a maioria dos professores são viciados e não conseguem viver sem essas maravilhas modernas), bebedouro, uma caixa com bolachas, três ventiladores velhos, alguns quadros de avisos e um grande conjunto de armários com cadeados de senhas.

Todos os dias, antes das suas aulas, o professor Papadopoulos passa na sala dos professores e pega o material para a aula do dia. Para pegar o seu material, ele precisa selecionar a sua senha corrente no cadeado do armário. Anualmente a administração da universidade pede para os professores limparem os seus armários e redefinirem a senha. A cada vez que o professor Papadopoulos vai abrir o armário, ele fica imaginando qual seria a melhor maneira de abrir seu cadeado com senha de acordo com a configuração atual.

Depois de algumas tentativas de identificar a melhor maneira de abrir o cadeado, o professor decidiu pedir ajuda aos alunos da Computação para definir um algoritmo para resolver o problema. O problema consiste em encontrar a menor sequência de rotações que devem ser realizadas para cada dígito do cadeado para que no final a configuração do cadeado seja igual a da senha. Caso haja um empate no número de rotações (o mesmo número de rotações tanto para cima quanto para baixo), a rotação para cima deve ser escolhida.

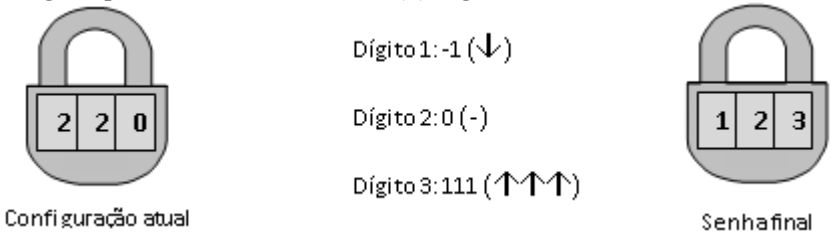
Entrada

Existem vários casos de teste, cada um contendo: um número N, $2 \leq N < 10$, de dígitos da senha do cadeado, seguido por dois conjuntos de N números indicando a senha que abre o cadeado e a configuração corrente de cada dígito do cadeado. Entre N, a senha e a configuração atual dos dígitos, existe um espaço em branco. Já, entre cada dígito, não existe espaço algum. Cada dígito do cadeado pode estar no intervalo de 0 a 9. Todos os parâmetros correspondem a números inteiros positivos. Um valor de N igual a zero (0) indica o fim das entradas.

Saída

Para cada caso de teste, você deve imprimir uma linha contendo N grupos de sequências de números. Deve haver um espaço entre cada grupo. Cada sequência pode conter X números -1 (↓), caso o dígito deva ser rolado X vezes para baixo, Y números 1 (↑), caso o dígito deva ser rolado Y vezes para cima. Caso o dígito não deva ser rolado, o grupo deve conter o valor 0 (zero).

Veja o exemplo a seguir, para um cadeado de três (3) dígitos. A entrada seria: 3 123 220.



A saída, por sua vez, deve conter 3 grupos de sequências, indicando quantas e quais rolagens devem ser realizadas no primeiro, segundo e terceiro dígitos, respectivamente. Para o exemplo, a saída correta seria: -1 0 111, indicando que o primeiro dígito deve ser rolado uma vez para baixo, o segundo dígito não deve ser alterado e o terceiro de ser rolado três vezes para cima. Não devem ser impressos espaços no fim de cada linha.

Exemplos

Exemplo de entrada

```
6 911007 312914
3 249 672
4 1111 3209
8 12345678 22042013
2 05 50
0
```

Saída para o exemplo de entrada

```
-1-1-1-1 0 -1 1 -1 111
-1-1-1-1 -1-1-1 -1-1-1
-1-1 -1 1 11
-1 0 111 0 111 -1-1-1-1 -1-1-1-1 11111
11111 11111
```

Solução

O problema é relativamente simples. Devemos só estar atentos às comparações que devem ser realizadas (sequência de ifs) e operações aritméticas simples.

Após a leitura dos dados, para cada entrada, devemos realizar a subtração da senha para a posição corrente, que indica quantas rotações teremos que realizar. Se o módulo deste valor for:

- zero, imprimir 0.
- 5, imprimir o número 1 (cima) cinco vezes.
- maior que 5, indica que temos que "dar a volta" no dígito. Para isso, devemos descobrir o maior e o menor número entre a senha e o número corrente. O número de rotações é o módulo da subtração entre o maior e o menor somado com 10.
    - Para descobrir a direção da rotação, devemos verificar se a subtração da senha para a posição corrente é positiva, então devemos rotacionar para baixo (-1), caso contário para cima (1);
    - Caso contrário (entre 0 e cinco), devemos apenas verificar a direção. Para descobrir a direção da rotação, devemos verificar se a subtração da senha para a posição corrente é positiva, então devemos rotacionar para cima (1), caso contário para baixo (-1);

Depois de descoberto o número de rotações e a direção é só executar um laço até o número de rotações e imprimir a direção.

Problema E

Filtragem de Imagens

Arquivo fonte: `filtragem.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Hericlapiton da Silva está escrevendo um sistema de processamento e análise de imagens digitais para a empresa de segurança de seus pais, que permite o reconhecimento de placas de veículos em imagens monocromáticas – obtidas a partir de câmeras de condomínio, por exemplo.

Uma imagem monocromática pode ser entendida como uma função que mapeia posições (x, y) para intensidades de brilho, representadas geralmente por meio do intervalo que vai de 0 a 255. Em imagens em níveis de cinza, 0 representa preto e 255 representa branco.

Uma etapa essencial do sistema que Hericlapiton está desenvolvendo é o Pré-Processamento, na qual a qualidade da imagem é melhorada por meio da redução de aspectos indesejáveis, como ruídos, ou o realce de características relevantes, como bordas.

Entre as técnicas empregadas nesta etapa está a filtragem no domínio espacial, que envolve o deslocamento de uma matriz, conhecida como máscara, por toda a imagem e a realização de operações que resultam em novos valores para os pixels.

A máscara, uma matriz com dimensão geralmente ímpar, tem o seu elemento central posicionado em cada um dos pixels da imagem. Quando centralizada em um pixel, o valor de cada célula da matriz é multiplicado pelo valor do pixel que está sob a célula. Os resultados destas multiplicações são então somados e o valor obtido é atribuído como novo valor do pixel sobre o qual a matriz foi centralizada.

No exemplo seguinte, está sendo aplicada uma máscara 3x3 em uma imagem monocromática também 3x3 pixels.

Imagem:

137	115	153
177	213	103
115	182	158

Máscara:

0,025	0,1	0,025
0,1	0,5	0,1
0,025	0,1	0,025

Aplicação da máscara na imagem (passo-a-passo/pixel-a-pixel):

f(1,1) = 103 (103,025)

	137	115	153
	177	213	103
	115	182	158

f(1,2) = 114 (114,80)

137	**115**	153	
177	213	103	
115	182	158	

f(1,3) = 103 (103,625)

137	115	**153**	
177	213	103	
115	182	158	

f(2,1) = 142 (142,425)

	137	115	153
	177	213	103
	115	182	158

f(2,2) = 178 (178,275)

137	115	153	
177	**213**	103	
115	182	158	

f(2,3) = 111 (111,325)

137	115	153	
177	213	**103**	
115	182	158	

f(3,1) = 98 (98,725)

	137	115	153
	177	213	103
	115	182	158

f(3,2) =146 (146,60)

137	115	153	
177	213	103	
115	**182**	158	

f(3,3) = 112 (112,825)

137	115	153	
177	213	103	
115	182	**158**	

Imagem resultante:

103	114	103
142	178	111
98	146	112

No exemplo apresentando, o resultado da aplicação da máscara foi arredondado para baixo. Como a máscara pode conter qualquer valor, pode ocorrer de o resultado da aplicação da máscara a um pixel ultrapassar o limite válido e retornar um valor menor que 0 ou maior que 255. Neste caso, uma estratégia adotada é o truncamento: todo valor maior que 255 é considerado igual a 255 e todo valor menor que 0 é considerado igual a 0. Ademais, quando parte da máscara ficou posicionada fora da imagem (o que acontece quando ela está centralizada nos pixels da extremidade), esta parte foi apenas desconsiderada no resultado final.

Como exemplo de cálculo, o valor da primeira célula da matriz resultante foi calculado a partir de: 0,025*0 + 0,1*0 + 0,025*0 + 0,1*0 + 0,5*137 + 0,1*115 + 0,025*0 + 0,1*177 + 0,025*213 = 103.

Auxilie Hericlapiton da Silva a implementar um algoritmo de filtragem no domínio espacial que recebe uma imagem e uma máscara, e aplica a máscara na imagem, calculando o resultado.

Entrada

A entrada é composta por diversos casos de teste. Cada caso de teste é iniciado por uma linha que contém quatro inteiros M, $1 \leq M \leq 500$, N, $1 \leq N \leq 500$, L, L é ímpar, $1 \leq L \leq 19$, e C, C é ímpar, $1 \leq C \leq 19$, separados por espaço, representando o número de colunas da imagem, o número de linhas da imagem, o número de linhas da máscara e o número de colunas da máscara, respectivamente. As N linhas seguintes contêm M inteiros separados por espaço representando os valores dos pixels de cada linha da imagem. As L linhas seguintes contém C reais com três casas decimais, separados por espaço, representando os valores de cada célula da linha da máscara. O fim da entrada é representado por uma linha contendo quatro zeros separados por espaço.

Saída

Para cada caso de teste, imprima N linhas contendo M inteiros cada, representando o resultado da aplicação da máscara na imagem. Trunque os resultados, se necessário. Como pixels só podem assumir valores inteiros, caso o resultado da aplicação da máscara resulte em um número real, arredonde para baixo. Não deve haver espaços no fim da linha.

Exemplos

Exemplo de entrada

```
3 3 3 3
137 115 153
177 213 103
115 182 158
0.025 0.100 0.025
0.100 0.500 0.100
0.025 0.100 0.025
0 0 0 0
```

Saída para o exemplo de entrada

```
103 114 103
142 178 111
98 146 112
```

Solução

Este é um algoritmo muito comum na área de Processamento e Análise de Imagens Digitais e pode ser implementado por meio de quatro estruturas de repetição aninhadas. As duas primeiras servem para passar por todos os pixels da imagem (1º: passe por todas as linhas; 2º: passe por todas as colunas) que

deve ser filtrada. Estando em um pixel I qualquer, as duas repetições internas devem passar pelos vizinhos de I compreendidos pela máscara de filtragem.

Para isso, podem ser calculados quantos pixels devem ser visitados à esquerda, à direita, acima e abaixo do pixel I – em função do tamanho da máscara. Como exemplo, para uma máscara 3x5 (3 linhas e 5 colunas), o algoritmo de filtragem deverá visitar os pixels desde 2 colunas antes de I até 2 colunas depois, desde uma linha acima de I até 1 linha abaixo, além do próprio pixel I. Para calcular estas distâncias, pode-se utilizar a seguinte fórmula: deltaLinha = piso(L/2) e deltaColuna = piso(C/2).

Portanto, a terceira repetição deve iterar desde I-deltaLinha até I+deltaLinha e a quarta repetição deve iterar desde C-deltaColuna até C+deltaColuna. Dentro das quatro repetições deve-se verificar se a posição atual é válida (está dentro da imagem) e, em caso afirmativo, realizar a multiplicação do pixel atual pelo correspondente na máscara.

Em seguida, deve-se verificar se o valor calculado não ultrapassa os limites válidos (0 e 255) e realizar o truncamento, se necessário.

Problema F

Piscinas

Arquivo fonte: `piscinas.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Cirinho recebeu a restituição do imposto de renda e está querendo construir uma piscina na sua casa de campo. Procurando evitar o convencional, ele deseja que a piscina tenha um formato diferente. Sua ideia é que o formato possa ser representado por vários retângulos dispostos em um plano. Alguns exemplos de formatos que ele aceitaria para a piscina estão representados a seguir:

A empresa que ele contratou para construir a piscina sabe construir apenas piscinas retangulares e, como os vendedores são ruins de matemática, não sabem calcular o custo total da piscina, dado o valor do m^2. Escreva um programa para ajudar Cirinho a descobrir o valor que ele pagará pela piscina, dadas as piscinas retangulares que serão utilizadas na sua construção e o custo do metro quadrado. Vale ressaltar que as áreas das piscinas que possuem intersecção são cobradas apenas uma vez pela empresa.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de cada caso de teste contém o número N, N > 0, de piscinas retangulares que formam a piscina de Cirinho. Cada uma das N linhas seguintes contém quatro inteiros x1, y1, x2, y2 (maiores ou iguais a zero), separados por espaço, representando as coordenadas x e y, no plano cartesiano, do canto superior esquerdo e inferior direito de cada piscina retangular que será utilizada na construção da piscina de Cirinho. A última linha de cada caso de teste contém um número inteiro representando o custo do metro quadrado da piscina. Assume-se que cada unidade do plano cartesiano representa 1 metro.

Saída

Para cada caso de teste, imprima uma linha contendo um número inteiro representando o custo total da piscina.

Exemplos

Exemplo de entrada

```
2
1 4 2 1
1 2 4 0
20
3
1 5 3 3
1 4 5 2
2 4 3 0
5
```

Saída para o exemplo de entrada

```
160
60
```

Solução

A solução deste problema envolvia encontrar as regiões de intersecção entre as piscinas e contá-las apenas uma vez. Para tanto, as piscinas podem ser ordenadas em relação ao canto esquerdo e, para cada piscina – desconsiderada a primeira -, pode-se verificar se o canto esquerdo dela está entre o início e fim de alguma piscina anterior. Se estiver, a área de intersecção pode ser contada. Outra forma de resolver o problema seria criar um plano cartesiano representado por uma matriz booleana utilizando alguma estrutura de dados dinâmica e, para cada piscina, atribuir o valor "true" para toda a área da piscina. Após fazer isso para todas as piscinas, bastaria contar o número de ocorrências de "true" para calcular a área total.

Problema G

Exploração de Planetas

Arquivo fonte: `naves.{c | cpp | java` *ou* `pas}`

Autor: Anderson Viçoso de Araújo (Fatec São José dos Campos)

Ano 2142. Naves espaciais e explorações de novos planetas são palavras comuns no dia-a-dia dos seres humanos. Apesar de toda a tecnologia existente, o capitão McNamara está precisando de uma ajuda para resolver um problema que tem se deparado diariamente no cumprimento do seu dever. Por não se tratar de um problema muito complexo, e como ele está um pouco enferrujado na programação de computadores (que após o domínio dos robôs inteligentes se tornou um mero passatempo para os seres humanos), decidiu pedir ajuda a você para implementá-lo.

Podem existir diversas naves espaciais viajando juntamente com o capitão a cada nova jornada - ele nunca viaja sozinho. Ao chegarem próximos de um planeta desconhecido, o capitão McNamara e a sua equipe se deparam sempre com o mesmo problema. Qual nave deve ser a primeira a entrar na atmosfera do planeta desconhecido? Após um consenso, ficou decidido que a nave mais próxima do novo planeta deveria iniciar o reconhecimento do mesmo, e assim por diante. Desta forma, o problema consiste em identificar a lista ordenada das naves espaciais do esquadrão partindo da mais próxima até a mais distante do planeta a ser explorado. Assim, temos como entrada: as coordenadas do planeta alvo, o número de naves existentes e as coordenadas de cada uma das naves. Caso haja um empate na distância entre as naves, escolha a primeira nave na ordem de entrada.

Entrada

Existem vários casos de teste, cada um contendo: três coordenadas (X, Y, Z) para o planeta a ser explorado, o número (N >= 2) de naves espaciais e um conjunto de tamanho N com as posições das naves espaciais onde cada posição contém três coordenadas. Todos os parâmetros estão separados por um espaço em branco e correspondem a números inteiros positivos. A última linha da entrada é especificada pelo número -1.

Saída

Para cada caso de teste, você deve imprimir uma linha indicando a lista ordenada por distância com os números das naves espaciais (os números são contados a partir de 0).

Exemplos

Exemplo de entrada

```
5 5 5 2 1 1 1 3 3 3
2 1 0 5 1 2 8 6 3 8 7 3 1 2 2 4 9 6 0
2 0 0 4 1 1 2 2 2 3 1 0 0 1 2 3
10 35 29 3 12 34 75 32 49 01 12 12 12
2 2 2 2 1 1 1 3 3 3
-1
```

Saída para o exemplo de entrada

```
1 0
3 2 0 1 4
2 0 1 3
2 1 0
0 1
```

Solução

O problema consiste em identificar a distância euclidiana entre as naves e o planeta de destino. Para fazer isso usamos a fórmula:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$

Onde 1, 2 e 3, indicam as coordenadas x, y e z.
Depois de calculadas e armazenadas as distâncias, devemos ordenar o vetor de acordo com a menor distância usando qualquer algoritmo de ordenação e apresentar o vetor final.

Problema H

Você é quadrado!

Arquivo fonte: `quadrado.{c | cpp | java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Zequinha é um menino muito esperto, mas é levado e muito falante. Ele falou demais na aula de matemática, até que a professora cansou de chamar a sua atenção, mas não adiantou. Virava e mexia e o Zequinha estava falando com alguém na hora errada. Resultado: castigo! Vai ter que fazer uma imensa lição extra de geometria no fim de semana que é, justamente, o fim de semana do campeonato de futebol do time do qual ele é o centroavante.

A lição até que é fácil, mas é meio trabalhosa: diversas medidas são dadas e Zequinha deverá indicar aquelas que podem formar um quadrilátero e aquelas que não podem. Zequinha aprendeu (ou deveria ter aprendido) que um quadrilátero possui quatro lados, onde a soma dos três lados menores é maior que o lado mais comprido, que se o quadrilátero tem os quatro lados iguais ele é um quadrado e se ele possui dois lados com uma mesma medida e outros dois lados com uma outra mesma medida, trata-se de um retângulo. Possivelmente a professora deve ter explicado mais alguma coisa, mas agora já era. Desesperado, Zequinha pediu pra você, seu querido tio craque em programação, fazer um programa que recebe vários casos com quatro medidas e indica o resultado correspondente, o que vai ajudar a salvar o fim de semana do sobrinho artilheiro falante.

Entrada

O conjunto de entradas se inicia com um inteiro N, $0 \leq N \leq 10000$, indicando a quantidade de casos a serem testados. Seguem-se então N linhas, cada uma contendo 4 inteiros A, B, C e D, $1 \leq A, B, C, D \leq 1000$, separados por um espaço, indicando as medidas a serem consideradas naquele caso.

Saída

Para cada caso de teste lido da entrada, você deverá imprimir uma linha contendo uma única palavra em minúsculas e sem acentos indicando o diagnóstico para aquele conjunto de medidas. Se não for possível construir qualquer quadrilátero com aquelas quatro medidas, imprima `invalido`; se tratar-se de um quadrado, imprima `quadrado`; se for um retângulo, imprima `retangulo`. Nos outros casos, imprima `quadrilatero`. Sabemos que um quadrado é também um retângulo e todo retângulo é também um quadrilátero, mas, neste problema, você deve imprimir apenas a palavra mais específica para cada caso.

Exemplos

Exemplo de entrada

```
4
12 12 12 12
90 10 10 90
1 2 3 4
157 35 26 60
```

Saída para o exemplo de entrada

```
quadrado
retangulo
quadrilatero
invalido
```

Solução

Bastava receber os 4 valores e compará-los. A solução mais simples seria primeiro descartar os casos de resposta `impossivel`, depois os de resposta `quadrado` e por fim as duas possibilidades restantes.

Problema I

Ah, esses vizinhos ...

Arquivo fonte: `vizinho.{c | cpp | java ou pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Um problema clássico em computação é o do caixeiro viajante (TSP – *Traveling Salesman Problem*), que é de fácil compreensão, mas oferece grandes dificuldades para ser resolvido. Um caixeiro viajante (um tipo de vendedor ambulante) deve visitar *n* cidades (C_1, C_2, C_3, ..., C_n), partindo da cidade C_1, passando uma única vez em cada cidade e voltando finalmente à cidade C_1. As distâncias d_{ij} entre qualquer par de cidades *Ci* e *Cj* são conhecidas. O problema consiste em determinar a sequência em que as cidades devem ser visitadas de maneira que a distância total percorrida seja a menor possível.

Como a forma geral desse problema pode comportar algumas particularidades, o único algoritmo conhecido que garante a melhor solução para qualquer instância válida do problema é o da força bruta, ou seja, requer computar todos os trajetos possíveis, calcular a distância total percorrida em cada um deles e escolher aquele que tiver o menor total. Isso é uma dificuldade, pois a menos que o número de cidades envolvidas seja muito reduzido, demandará um volume imenso de processamento, em alguns casos proporcionando computações que levarão anos ou séculos para serem concluídas, mesmo utilizando computadores extremamente rápidos.

Para problemas como esse, em geral, são utilizados métodos denominados heurísticos, que diminuem o tempo de computação por simplificar de alguma forma o processamento, mas que não garantem a obtenção do melhor resultado possível, porém geralmente produzem soluções aceitáveis. Uma heurística que pode ser utilizada para o problema do caixeiro viajante é a do Vizinho Mais Próximo. Consiste em fazer com que a próxima cidade seja sempre aquela mais próxima da cidade atual e que ainda não foi visitada. Obviamente, após a última cidade ser visitada, fazemos o retorno para a cidade inicial do percurso, para concluir o ciclo. Dependendo das características do conjunto de cidades e de suas distâncias, essa heurística vai, frequentemente, produzir soluções ótimas, a um custo muito baixo. É claro que, em outros casos, ela pode produzir soluções não tão boas, ou mesmo ruins.

Seu objetivo, neste problema, é verificar se a solução fornecida por essa heurística corresponde ou não à melhor solução para um conjunto de cidades apresentado na entrada.

Entrada

A entrada é iniciada por um valor *N*, $0 \leq N \leq 50$, que indica a quantidade de casos de teste a serem processados. Cada um dos *N* casos de teste inicia-se com um inteiro *V*, $2 \leq V \leq 15$, que indica a quantidade de cidades a serem visitadas, um inteiro *A*, $V\text{-}1 \leq A \leq V(V\text{-}1)/2$, que representa a quantidade de trajetos a serem informados entre as cidades, e um inteiro *T*, $T \leq 2^{32}\text{-}1$, que corresponde à menor distância possível para aquele conjunto de cidades. Seguem-se *A* linhas, cada uma contendo três inteiros separados por um espaço em branco, onde os dois primeiros são duas cidades e o terceiro é a distância entre elas. Assuma que a distância para ir de uma cidade *X* para uma cidade *Y* é sempre a mesma que para ir da cidade *Y* para a cidade *X*.

Saída

Para cada caso de teste, imprimir uma linha contendo 'S' (em maiúsculas, sem aspas) caso a heurística do Vizinho Mais Próximo produza a melhor solução e 'N' (em maiúsculas, sem aspas), caso contrário.

Exemplos

Exemplo de entrada

```
2
6 15 203
0 1 42
0 2 20
0 3 31
```

```
0 4 29
0 5 33
1 2 58
1 3 45
1 4 68
1 5 47
2 3 28
2 4 32
2 5 51
3 4 57
3 5 61
4 5 34
5 9 158
0 1 18
0 2 20
0 3 70
0 4 80
1 2 15
1 4 35
2 3 36
2 4 60
3 4 50
```

Saída para o exemplo de entrada

```
S
N
```

Solução

É um problema básico de percurso em grafos ponderados. Como a quantidade de vértices e arestas é pequena, o grafo pode ser representado por matriz de adjacências. A entrada já informa qual é a melhor solução possível, que foi calculada previamente por um algoritmo de forca bruta. A solução precisa apenas implementar um algoritmo guloso, partindo do vértice zero (sempre) e indo para o seu vizinho mais próximo ainda não visitado. Deste, segue-se para o vizinho mais próximo ainda não visitado. Do ultimo vértice atingido, faz-se o retorno para o vértice zero. Somar os pesos de todas as arestas percorridas. Se for igual ao valor da melhor solução daquele grafo que foi lida na entrada, imprime 'S', senão, imprime 'N'.

Problema J

Derivadas

Arquivo fonte: `derivadas.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Derivadas são importantes ferramentas do Cálculo que representam a taxa de variação instantânea de uma função. Por meio delas, é possível mensurar como uma função varia quando seus parâmetros de entrada variam. Um exemplo é a velocidade, derivada da função de posição de um objeto em relação ao tempo.
A derivada de uma função y=f(x) é geralmente representada por f'(x).

Existem algumas regras básicas que facilitam a obtenção da derivada de uma função y=f(x). A seguir, três dessas regras estão listadas:

1. A derivada de uma constante é igual a 0.
2. A derivada de $x^n = n.x^{n-1}$.
3. A derivada de uma soma/subtração é igual à soma/subtração das derivadas de cada termo.

A seguir, são apresentados exemplos de derivadas, obtidas a partir destas regras, para algumas funções:

- y=f(x)=2 f'(x)=0
- y=f(x)=2x^3 f'(x)=6x^2
- y=f(x)=2x^3+2 f'(x)=6x^2
- y=f(x)=2x^3+3x^3 f'(x)=15x^2
- y=f(x)=2x^3-4x^2 f'(x)=6x^2 - 8x

Ednaldo, seu professor de cálculo, precisa da sua ajuda para escrever um programa que, dada uma função y=f(x), retorna a derivada desta função. A função y pode conter apenas constantes inteiras, somas, subtrações e potências com base x. A derivada obtida pelo seu programa deve simplificar as expressões obtidas, somando/subtraindo termos que possam ser combinados. Ademais, as potências com base x devem ser exibidas em ordem decrescente de expoente.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste possui a fórmula de uma função que contém constantes inteiras positivas e negativas, somas, subtrações e potências com base x. Entre os termos da equação podem existir espaços ou não.

Saída

Para cada caso de teste, imprima uma linha contendo a fórmula simplificada da derivada da função. Assuma que, entre os termos e sinais de soma e subtração, deve ser impresso um espaço.

Exemplos

Exemplo de entrada

```
x^2
2x^3
2+3x^4+2x^3
2x + 4x^2 + 5x^2 + 10x
x + 3x^2 + 5x^2 + 5x + 3x^3
```

Saída para o exemplo de entrada

```
2x
6x^2
12x^3 + 6x^2
18x + 12
9x^2 + 16x + 6
```

Solução

Diferentemente do que alguns alunos comentaram em Clarifications, este não se trata de um problema de Matemática, mas sim de processamento de Strings. Existem diferentes estratégias para resolver o problema. Uma delas envolveria utilizar algum método que "quebra" Strings, como o split do Java. Outra envolveria processar a String caractere a caractere até que fosse encontrado um x ou um sinal ("+" ou "-"). No caso do "x", tudo, desde o último expoente de x poderia ser considerado o coeficiente e, caso o x fosse seguido do caractere ^, o que vier depois, mas antes dos sinais de "-" ou "+" poderia ser considerado expoente.

Para cada potência de x, deveria ser acumulado o coeficiente e, depois tudo impresso em ordem decrescente de potência. Uma forma de implementar isso seria por meio de um mapa, no qual a chave é a potência de x (iniciando em 0) e o valor é o coeficiente associado àquela potência. Valores do mapa com coeficientes iguais a 0 deveriam ser descartados.

Problema K

Primos gêmeos

Arquivo fonte: `gemeos.{c | cpp | java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Números primos são um grande tema de pesquisa na Matemática. Para quem não se lembra, um número primo é aquele que é divisível apenas por ele mesmo e pela unidade. Por definição, o 1 não é primo. Acredita-se que existam infinitos números primos, que se distribuem de forma bastante variada ao longo do eixo dos inteiros. Um caso particular é o dos chamados primos gêmeos, que são pares de números primos separados entre si por apenas um inteiro, como, por exemplo, o 5 e o 7, ou então o 11 e o 13. Neste problema você deverá determinar quantos pares de primos gêmeos existem dentro de um determinado intervalo.

Entrada

A entrada é iniciada por um valor N, $0 \leq N \leq 100$, que indica a quantidade de intervalos a serem testados. Seguem-se N linhas compostas por pares de inteiros A e B separados por um espaço em branco, com $A \leq B$, $1 \leq A, B \leq 1.000.000$, representando os limites do intervalo a ser considerado.

Saída

Para cada intervalo informado, imprimir uma linha contendo um inteiro Q indicando a quantidade de pares de primos gêmeos ali encontrados.

Exemplos

Exemplo de entrada

```
3
1 20
101 199
500 510
```

Saída para o exemplo de entrada

```
4
7
0
```

Solução

Este problema requer que se determine o conjunto de números primos de um intervalo, verificando os pares em que a diferença entre os primos é igual a dois. Entretanto, como são várias consultas a serem processadas, calcular os primos do intervalo a cada caso de teste torna a solução muito lenta (Time Limit Exceeded). Para que o algoritmo seja suficientemente rápido para resolver todas as consultas dentro do tempo limite é necessário primeiro montar uma tabela com todos os primos até um milhão (o limite superior previsto) e então processar as consultas, percorrendo a tabela de primos e contando os primos gêmeos existentes naquele intervalo – ou, o que seria melhor, montar um vetor com a quantidade acumulada de primos gêmeos no intervalo e apenas consultar esse vetor para cada entrada. Os casos de teste previam intervalos válidos bem maiores que os do exemplo, o que pode ter feito algumas equipes acreditarem que possuíam uma solução rápida quando de fato não a tinham.

Problema L

Prioridades

Arquivo fonte: `nihans.{c | cpp | java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Ermengarda trabalha na assessoria da direção de um grande hospital particular. Em tempos extremamente competitivos como os atuais, é importante manter um bom relacionamento com os clientes, pois a percepção destes quanto ao serviço oferecido tem impactos muito sérios na viabilidade do negócio.

O hospital implementa um sistema que coleta críticas, sugestões e elogios dos clientes por meio de questionários em papel, caixas de sugestões e uma página na internet, com o propósito de saber qual a percepção que o público atendido tem da empresa. Ermengarda precisa fazer um relato dos principais problemas apontados pelo público e para isso reuniu todas as respostas e as separou em categorias como Preços, Qualidade do Atendimento, Conforto, Estacionamento, Alimentação, etc.

Para cada categoria, foram totalizadas quantas reclamações ela teve e agora será preciso separá-las em três classes de prioridades, em que a classe A representa a prioridade mais alta, a classe B representa a prioridade média e C a prioridade baixa. Uma categoria que possua um número muito alto de reclamações terá uma prioridade maior que uma categoria que possua um número significativamente menor de reclamações. O problema agora é definir o critério para fazer essa separação. Existe, é claro, o método de Pareto para tratar o problema, mas Ermengarda ouviu falar de uma outra abordagem, chamada índice de Nihans, que lhe pareceu bem interessante.

Primeiro, é calculado um valor, o tal índice de Nihans, cujo cálculo é bem simples: soma-se o quadrado dos valores e divide-se pela soma desses mesmos valores. Por exemplo, vamos supor que Ermengarda possua 10 categorias, e os totais de reclamações de cada uma seja 18, 23, 72, 9, 5, 28, 93, 34, 52 e 71. O índice é a soma dos quadrados desses números, que dá 24477, dividida pela soma desses números, que é 405, o que resulta em 60.4 aproximadamente.

Então, todas as categorias cuja quantidade de reclamações seja superior a 60.4 serão consideradas como de prioridade A, ou seja, as categorias que tiveram 93, 72 e 71 reclamações neste exemplo. Em seguida, repete-se o procedimento anterior para calcular um novo índice, que será usado para separar os itens restantes nas classes B e C. Em nosso exemplo, seriam somados os quadrados de 18, 23, 9, 5, 28, 34 e 52 (que são os itens que não estão na classe A) e dividimos esse valor (que, por acaso é 5603) pela soma desses mesmos valores (que dá 169). O novo índice é, portanto, 33.2 o que significa que na classe B teremos as categorias cujas quantidades de reclamação são 34 e 52. As categorias restantes pertencem à classe de prioridade C.

Ermengarda pediu a você, estagiário de informática no hospital, para escrever um programa capaz de dividir um conjunto de números nas classes de prioridade A, B e C, conforme o índice de Nihans.

Entrada

A entrada é iniciada por um valor N, $0 \leq N \leq 50$, que indica a quantidade de casos de teste a serem processados. Cada um dos N casos de teste é apresentado em uma linha da entrada, iniciando por um inteiro Q, $3 \leq Q \leq 1000$, que indica a quantidade de reclamações para cada categoria a ser classificada. Seguem-se então Q inteiros separados por um espaço em branco.

Saída

Para cada caso de teste, imprima uma linha indicando o número do caso de teste, conforme o exemplo a seguir, e depois três linhas, a primeira com os números que pertencem à classe A, a segunda com aqueles que pertencem à classe B e outra linha com os números da classe C. Os números devem ser impressos na ordem em que aparecem na entrada. Deixe uma linha em branco após cada caso de teste.

Exemplos

Exemplo de entrada

```
2
10 18 23 72 9 5 28 93 34 52 71
20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Saída para o exemplo de entrada

```
Caso 1
A: 72 93 71
B: 34 52
C: 18 23 9 5 28

Caso 2
A: 14 15 16 17 18 19 20
B: 10 11 12 13
C: 1 2 3 4 5 6 7 8 9
```

Solução

O problema requer que se calcule um valor real a partir de um conjunto de inteiros e separar como classe ‘A’ os números previamente informados que são maiores ou iguais a esse real. Em seguida, repete-se o mesmo processo para os números que sobraram e aqueles que forem maiores que esse segundo real, pertencem à classe ‘B’, sendo os restantes classe ‘C’. Do ponto de vista da lógica, é um problema muito simples, sendo o principal problema encontrado pelas equipes ligado à formatação da saída: deve haver um espaço em branco antes de cada inteiro a ser impresso, mas não pode haver um espaço em branco após o último inteiro da linha. Uma linha em branco precisa ser deixada entre a resposta de um caso e outro.

Problema M

Coisa de nerd ...

Arquivo fonte: `placas.{c | cpp | java` *ou* `pas}`

Autor: Antonio Cesar de Barros Munari (Fatec Sorocaba)

Argemiro é um nerd meio bitolado em computação. Gosta de programação, computadores, redes, RPG, HQ, o kit completo. Como vai comprar um carro em breve, tem começado a prestar atenção nos diversos modelos de veículos que passam à sua frente no trânsito infernal de sua cidade. Estes dias, ele percebeu uma coisa curiosa: um carro que a placa era IOI 1010. Isso mexeu com o coração nerd de Argemiro, pois era uma placa que podia ser vista como um número binário (1011011, ou seja, 91 em base binária). Nosso herói agora vive prestando atenção nas placas de carro, buscando aquelas que se configuram como placas binárias, ou seja, compostas apenas pelas vogais I e O e os dígitos numéricos 0 e 1.

Sua tarefa é, para uma determinada placa avistada por Argemiro, decidir se ela é uma placa binária ou não.

Entrada

O conjunto de entradas se inicia com um inteiro N, $0 \leq N \leq 10000$, indicando a quantidade de casos a serem testados. Seguem-se então N linhas, cada uma contendo uma *string* composta por exatamente 7 caracteres, sendo os 3 primeiros letras maiúsculas sem acento ou cedilha e os 4 seguintes, dígitos numéricos, representando uma placa de carro.

Saída

Para cada caso de teste lido da entrada, você deverá imprimir uma linha contendo uma única palavra: NERD! (em maiúsculas seguida por uma exclamação) se a placa for binária e Normal (inicial maiúscula e restante em minúsculas) se a placa não for binária.

Exemplos

Exemplo de entrada

```
5
ABC1234
OOO0000
OIO0102
OOO0001
QIQ1111
```

Saída para o exemplo de entrada

```
Normal
NERD!
Normal
NERD!
Normal
```

Solução

Problema simples de verificação de *strings*. Receber a placa e verificar se os caracteres presentes restringem-se apenas às letras ‘I’ e ‘O’ e aos dígitos ‘1’ e ‘0’. Em caso afirmativo, imprimir uma linha com a *string* ‘Normal’, senão imprimir uma linha contendo ‘NERD!’, respeitando a especificação sobre maiúsculas e minúsculas indicada no enunciado.

Problema N

Serviço de Atendimento Aéreo de Urgência

Arquivo fonte: `saau.{c | cpp | java` *ou* `pas}`

Autor: Leandro Luque (Fatec Mogi das Cruzes)

Sabaúna é um distrito muito grande e populoso. Apesar de contar com muitos serviços públicos de excelente qualidade, a população sabaunense anda descontente com o atendimento móvel emergencial do distrito. Durante os horários de pico, o tráfego de veículos é tão intenso que as ambulâncias não conseguem atender a chamadas em menos de 30 minutos, o que resulta, algumas vezes, em morte ou sequelas irreversíveis para os pacientes que deveriam receber atendimento rápido.

Procurando mudar esse cenário, o responsável pelo distrito, Danilo, está estudando a implantação de um Serviço de Atendimento Aéreo de Urgência (SAAU), utilizando helicópteros. Para tanto, ele precisa definir o modelo de helicóptero que comprará e a melhor localização para instalar a base onde os helicópteros ficarão. Existem muitos modelos diferentes de helicóptero, cada um com um raio possível de ação e um custo de combustível.

Escreva um programa de computador para ajudar Danilo a descobrir a posição de instalação da base, de tal forma que o raio de ação do helicóptero seja o menor possível e toda a população esteja coberta.

Entrada

A entrada é composta por vários casos de teste. Cada caso de teste inicia-se com um número N, $N \geq 5$, indicando a quantidade de coordenadas que serão informadas. Cada coordenada representa a posição no plano cartesiano de um local onde o helicóptero deve prestar atendimento. As N linhas seguintes contêm dois números inteiros X e Y, $1 \leq X \leq 5000$ e $1 \leq Y \leq 5000$, separados por um espaço, indicando a posição x e y de cada coordenada, respectivamente. O último caso de teste é seguido por uma linha com o número 0 (zero).

Saída

Para cada caso de teste, imprima uma linha contendo 2 números inteiros e um número real, separados por espaço, indicando a posição x e y no plano cartesiano onde a base deve ser instalada, e o raio de cobertura do helicóptero, de tal forma que todos os locais (coordenadas) especificados sejam atendidos e o raio seja o menor possível. O número real deve ser especificado com precisão de um ponto decimal.

Exemplos

Exemplo de entrada

```
5
10 20
30 40
5 15
3 100
3 25
10
540 200
453 125
343 400
245 245
138 565
565 138
300 300
400 400
500 500
450 450
0
```

Saída para o exemplo de entrada

```
4 57 42.5
351 351 301.9
```

Solução

Deveria ser implementado um algoritmo que encontra o menor círculo envolvente (Minimal Enclosing Disk/Circle). Existem vários algoritmos. Em termos de desempenho computacional, acredito que o melhor seja o de Wezl (1991), que resolve o problema em tempo linear. Para mais informações: (http://www.sunshine2k.de/coding/java/Welzl/Welzl.html).