PROVA TOPCOM 12

14 de Junho de 2014

Realização: Patrocínio: Apoio:

Problema A
Plágio Musical

Nome do arquivo fonte: **plagio.c, plagio.cpp ou plagio.java**

As notas musicais são unidades básicas da música ocidental tradicional. Cada nota está associada a uma frequência. Duas notas musicais cujas frequências fundamentais tenham uma relação de potência de 2 (uma metade da outra, uma duas vezes a outra, etc.) são percebidas como muito similar. Por isso, todas as notas com esse tipo de relação recebem o mesmo nome, como descrito a seguir.

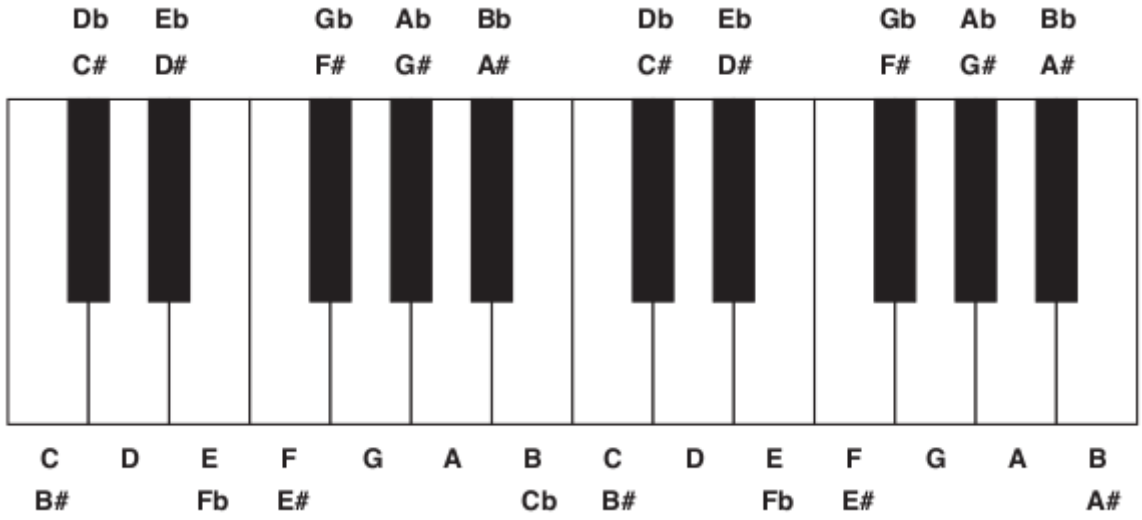
Há doze notas básicas, em uma sequência crescente de frequências, cada nota separada da anterior por uma mesma distância na escala musical (essa distância é chamada de semi-tom). Sete dessas doze notas são representadas por letras do alfabeto (A, B, C, D, E, F e G). A tabela abaixo mostra a distância, em semi-tons, entre essas notas.

Notas	A-B	B-C	C-D	D-E	E-F	F-G	G-A
Número de semi-tons	2	1	2	2	1	2	2

Note que há cinco notas que não são representadas pelas letras do alfabeto: as que estão entre A e B, entre C e D, entre D e E, entre F e G e entre G e A.

As notas podem ser modificadas por duas *alterações cromáticas*: *sustenido* e *bemol*, representadas respectivamente pelos símbolos '#' e 'b'. Sustenido altera a nota em meio tom para cima, e bemol altera a nota em meio tom para baixo. Uma nota com alteração cromática é denotada pelo nome da nota seguida pelo símbolo da alteração. Note que com esse esquema conseguimos representar todas as doze notas.

A figura abaixo ilustra o nome das notas, segundo o esquema descrito acima, em um trecho de teclado de piano.



Uma melodia pode ser representada por uma sequência de *notas musicais*. Por exemplo,

A A D C# C# D E E E F# A D G# A

é uma melodia muito conhecida. Note no entanto que, como as distâncias entre os semi-tons são sempre iguais, a mesma melodia pode ser escrita iniciando em outra nota (dizemos que a melodia está em outro tom):

B B E D# D# E Gb Gb Gb G# B E A# B

Sua vizinha é uma famosa compositora que suspeita que tenham plagiado uma de suas músicas. Ela pediu a sua ajuda para escrever um programa que, dada a sequência de notas da melodia de sua música, e a sequência de notas de um trecho de melodia suspeito, verifique se o trecho suspeito ocorre, em algum tom, na música dada.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de um caso de teste contém dois inteiros **M** e **T** ($1 \leq \mathbf{M} \leq 100000$, $1 \leq \mathbf{T} \leq 10000$, $\mathbf{T} \leq \mathbf{M}$), indicando respectivamente o número de notas da música e do trecho suspeito de ter sido plagiado. As duas linhas seguintes contêm **M** e **T** notas, respectivamente, indicando as notas da música e do trecho suspeito.

As notas em cada linha são separadas por espaço; cada nota é uma dentre 'A', 'B', 'C', 'D', 'E', 'F' ou 'G', possivelmente seguida de um modificador: '#' para um sustenido ou 'b' para um bemol.

O último caso de teste é seguido por uma linha que contém apenas dois números zero separados por um espaço em branco.

Saída

Para cada caso de teste, imprima uma única linha contendo um caractere: 'S' caso o trecho realmente tenha sido plagiado pela música ou 'N' caso contrário.

Exemplo:

Entrada	Saída
16 4 D G A B C D G G G C D E F# G C C G G C D 12 2 C C# D D# E F F# G G# A A# B C D 12 2 C Db D Eb E F Gb G Ab A Bb B C D 4 3 C E G Bb D F# A 0 0	S N N S

Problema B
Vinhos

Nome do arquivo fonte: **vinhos.c, vinhos.cpp ou vinhos.java**

Aproveitando sua viagem enóloga pela França, o professor Omerix trouxe algumas garrafas de vinhos tais como: beaujolais, tintos de Bordeaux, terroir da Auvergne, os nobres da Borgonha, rose da Provence, etc. Vinhos de diversas regiões e características. Ao chegar no Brasil, Omerix se deparou que cada vinho tinha uma temperatura mínima e máxima para sua conservação ideal. Sua tarefa é ajudar o professor Omerix a escolher a temperatura ideal para sua secreta adega (agora não mais secreta...), de forma a cobrir uma faixa com a maior quantidade possível de vinhos. Claro, o prof. Omerix está preocupado com sua conta de energia elétrica e a sustentabilidade do planeta.

Entrada

A entrada é composta por vários casos de testes. Cada caso de teste é iniciado por uma linha com um valor **n** que indica a quantidade de vinhos a serem avaliados. Cada uma das n linhas seguintes é composta por dois valores: $t_{\min}$ e t_{max}, a temperatura mínima e a máxima para uma boa conservação do vinho. Sabe-se que a adega do professor Omerix aceita temperaturas inteiras que seguem de -10 graus até 20 graus Celsius, pois atende aos vinhos que não dão dor de cabeça no dia seguinte. Resumindo: $-10 \leq t_{\min}$ e $t_{max} \leq 20$ e $1 \leq n \leq 1000$. A entrada termina com n = 0.

Saída

Para cada caso de teste imprima uma linha com um inteiro que represente a maior temperatura que contemple o maior número de vinhos. Caso não exista uma temperatura comum que esteja presente em dois ou mais vinhos, imprima a temperatura t_{max} do vinho que tenha o menor valor de $t_{\min}$ (Uma chance de preservar os vinhos por temperatura baixa, dado o nosso clima tropical). Veja os exemplos dos casos de testes. Para n = 0, não imprima nada.

Exemplo:

Entrada	Saída
3 -3 5 -1 10 9 15 2 3 5 7 15 5 1 3 2 4 3 5 4 6 1 10 0	10 5 4

Problema C

Difícil de Acreditar, mas Verdade!

Nome do arquivo fonte: **mas_verdade.c, mas_verdade.cpp ou mas_verdade.java**

A luta continua na decisão sobre armazenar números iniciados por seus dígitos mais
significativos ou menos significativos. Às vezes isto é chamado de "Endian War". O campo
de batalha remonta aos períodos do início da Ciência da Computação. Joe Stoy, em seu (a
propósito excelente) livro "Denotational Semantics", nos conta a seguinte estória:

"A decisão sobre em que direção os dígitos são dispostos é, naturalmente, matematicamente trivial. De fato, um computador britânico antigo apresentava números da direita para a esquerda (porque o cursor num osciloscópio funcionava da esquerda para a direita, mas em lógica serial os dígitos menos significativos eram processados antes). Turing costumava assombrar a audiência em suas palestras quando, quase por acidente, ele se utilizava desse sistema e escrevia coisas como 73+42=16. A versão seguinte da máquina foi construída de forma mais convencional simplesmente trocando os fios de deflexão no eixo X: o que, no entanto, preocupava os engenheiros pois seus gráficos de onda apareciam invertidos. Este problema foi por sua vez resolvido através de uma pequena janela onde os engenheiros (que geralmente costumavam estar atrás do computador de qualquer forma) pudessem ver a tela do osciloscópio por trás. [C. Strachey - comunicação privada]."

Você fará o papel da audiência e julgará a veracidade das equações de Turing.

Entrada

A entrada contém vários casos de teste. Cada um especifica uma equação de Turing na forma **a + b = c**, onde **a**, **b**, **c** são números compostos por dígitos **0..9**. Cada número pode conter no máximo 7 dígitos. Isto inclui número iniciados ou seguidos de zeros. A equação "0+0=0" encerra a entrada e precisa ser processada também. Não existem espaços em branco no meio das equações.

Saída

Para cada caso de teste, imprima "True" ou "False" indicando se a equação for verdadeira
ou falsa, respectivamente, de acordo com a interpretação de Turing (os números sendo lidos invertidos).

Exemplo:

Entrada	Saída
73+42=16 5+8=13 10+20=30 0001000+000200=00030 1234+5=1239 1+0=0 7000+8000=51 0+0=0	True False True True False False True True

Problema D
Monte Crista

Nome do arquivo fonte: **monte.c, monte.cpp ou monte.java**

Nas proximidades de Joinville há o famoso caminho dos Peabiru, ou Tape Aviru - que ligava o Atlântico e o Pacífico. Resquícios desse caminho pré-hispânico são encontrados em pontos descontínuos nos estados de Santa Catarina, do Paraná, de São Paulo, no Paraguai, na Bolívia e no Peru. E pelo que dizem, a conhecida Trilha Inca faz parte desse caminho.

A trilha de acesso começa no morro Monte Crista. Os poucos remanescentes em pedra contam uma história notável: pessoas, cargas, sonhos partiam da região de São Francisco do Sul (ou de Iguape/Cananéia, no litoral paulista) e subiam a serra na direção do interior da América, articulando civilizações da costa atlântica às do altiplano andino e da costa pacífica.

Dizem que próximo ao acesso final deste altiplano há uma escadaria feita de pedras. Nestas pedras encontraram-se algumas inscrições, as quais codificavam mensagens entre os jesuítas que ali utilizavam a trilha.

Um grupo de ecologistas, em visita ao morro Monte Crista encontrou uma destas inscrições codificadas, e como havia um pesquisador/estudioso em códigos neste grupo, este decifrou o código. A euforia foi geral, a imprensa foi chamada, o *circo foi armado*, para saberem como os ecologistas tinham decodificado as inscrições.

De acordo com o pesquisador de código, o alfabeto jesuíta era composto pelas letras de A até Z (incluindo as letras e K, W e Y). A codificação era realizada da forma que segue. Inicialmente, era construída uma lista em que a letra A aparecia na primeira posição, a letra B aparecia na segunda, e assim sucessivamente, com as letras seguindo a mesma ordem que em nosso alfabeto. Em seguida, o texto a ser codificado era varrido da esquerda para a direita e, para cada letra l encontrada, o número de sua posição na lista era impresso e l (a letra) era movida para o início da lista. Por exemplo, a codificação jesuíta para a mensagem:

ABBBAABBBBACCABBAAABC

era dada pela seguinte sequência de inteiros:

12112121112312312 1123

Os ecologistas pediram a sua ajuda para construir um programa que receba uma sequência de inteiros que representa uma mensagem codificada e decodifica-a.

Aviso: você só deve ir lá no inverno, preservando a ecologia do local e preparado, pois um número significativo de cobras jararacas procuram ajudar a preservar este lugar.

Entrada

Seu programa deve estar preparado para trabalhar com diversas instâncias. Cada a instância tem a estrutura que segue. Na primeira linha fornecido um inteiro m ($0 \leq m \leq 10000$) que representa o número de inteiros que compõem o texto codificado. Na próxima linha são dados, separados por espaços em branco, os m valores inteiros (cada valor é maior ou igual a 1 e menor ou igual a 26).

Um valor m = 0 indica o final das instâncias e não deve ser processado. Mas a linha em branco já foi impressa!

Saída

Para cada instância solucionada, você dever imprimir um identificador Instancia h em que h é um número inteiro, sequencial e crescente a partir de 1. Na linha seguinte, você deve imprimir o texto decodificado. Uma linha em branco deve separar a saída de cada instância.

Exemplo:

Entrada	Saída
21 1 2 1 1 2 1 2 1 1 1 2 3 1 2 3 1 2 1 1 2 3 5 22 6 8 4 15 3 24 1 1 26 22 10 6 4 13 16 16 12 5 1 4 20 1 21 21 5 10 7 16 6 15 12 5 3 8 9 0	Instancia 1 ABBBAABBBBACCABBAAABC Instancia 2 VEGAN Instancia 3 XXX Instancia 4 VIDALONGAAOSSTRAIGHTEDGERS

Problema E
Sequências

Nome do arquivo fonte: **sequencias.c, sequencias.cpp ou sequencias.java**

Uma sequência aritmética apresenta um primeiro número, e então uma sequência de outros números com diferença fixa entre eles. Por exemplo: 3, 5, 7, 9 é uma sequência aritmética pois ela tem como primeiro número o 3. E então, cada um de seus termos da sequência é formado pela adição do valor 2 ao termo anterior. Ou seja, os termos são diferentes por 2.

O 3 é o primeiro termo ou termo 1, o 9 o quarto termo ou o termo 4. Dado um número de partida, uma diferença e um valor, seu programa deve indicar se este último número pode tomar parte da sequência. Se sim, a saída indicará qual o número do termo nesta sequência, e se não, a saída deverá ter a letra X.

Entrada

A entrada consistirá de um número de linhas, onde cada linha tem 3 números separados por espaços em branco. O primeiro número é um inteiro que é o primeiro termo da sequência. O segundo é a diferença, o qual será um inteiro diferente de zero. O terceiro termo é o valor o qual você necessitará testar para determinar se este número pode fazer parte da sequência ou não. A entrada é finalizada por um valor zero para cada um dos três números (0 0 0).

Saída

A saída consistirá de uma linha para cada entrada. Ela terá como o valor o número que indica a qual o termo ele é, ou X se o número não fizer parte da sequência.

Exemplo:

Entrada	Saída
3 2 11 -1 -3 -8 0 0 0	5 X

Explicando:

- No primeiro caso, o 11 é o quinto termo da sequência, ou o termo 5;
- No segundo caso, o -8 não faz parte da sequência.

Problema F

Ajude Carlinhos da EngComp!

Nome do arquivo fonte: **carlinhos.c, carlinhos.cpp ou carlinhos.java**

Autor: Leonardo Ferreira Meneses

Carlinhos é um jovem estudante de Engenharia de Computação da UFES e apesar de estar no 10° período, em 9 anos de curso, possui um grande problema: enlouqueceu. Uma de suas loucuras foi expressa na forma de uma síndrome, diagnosticada como a "Síndrome do Maior Caminho (SMC)". Os efeitos dessa síndrome são um tanto desagradáveis: dado um mapa de prédios ligados por estradas, Carlinhos necessita andar a distância equivalente ao par de prédios mais distantes sempre que chegar ao local, de forma que qualquer outro caminho deve ser descartado. Como Carlinhos decidiu morar na UFES até terminar seu Projeto de Graduação (PG), está sempre presente em locais com mapas bem simples que seguem sempre as restrições:

(a) todas as estradas são de mão dupla;
(b) todas as estradas possuem 1km de comprimento e, portanto, toda estrada ligando dois prédios tem o mesmo comprimento;
(c) toda estrada conecta apenas dois prédios;
(d) dados dois prédios quaisquer *A* e *B*, só existe uma única maneira de chegar em *A* partindo de *B*, e vice-versa.

Carlinhos teve a brilhante ideia de construir um programa que dado um mapa que segue as restrições acima, determina de forma rápida a distância do maior caminho que ele precisa percorrer.

Apesar de ter feito Programação I, Programação II, Programação III, Estrutura de Dados I e Estrutura de Dados II, Carlinhos não lembra muito bem como programar. Mas como ele é muito seu amigo, e sabendo da sua facilidade em programação, resolveu pedir sua ajuda para construir o programa.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de um caso de teste contém um inteiro N representando o número de prédios no mapa. Cada uma das $N-1$ linhas seguintes de um caso de teste contém dois inteiros A e B indicando que existe uma estrada entre os prédios A e B. Os casos de teste terminam com N = 0.

Saída

Para cada de teste imprima uma única linha da saída contendo um inteiro indicando a distância entre um par de prédios mais distantes.

Exemplo:

Entrada	Saída
10 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 5 1 2 2 3 3 4 3 5 0	9 3

Problema G
Polyana Okimoto

Nome do arquivo fonte: **polyana.c, polyana.cpp ou polyana.java**

Polyana é uma nadadora olímpica da modalidade "maratona em águas abertas". Como você, ela é uma maratonista e sabe que nada ocorre por acaso, mas sim com muito treino.

Nesta fase final de preparação para um torneio, ela precisa aumentar a sua velocidade e, para isto, ela precisa controlar a sua passagem a cada 100m. Assim, ela precisa saber o menor e o maior tempo de passagem a cada 100m, durante os seus treinos de média distância. Ajude o técnico de Polyana a dizer a ela quanto foi o melhor e o pior tempo nas passagens de 100m.

Entrada

O arquivo de entrada contém vários cenários. Cada entrada de teste começa com um valor m, tal que 1 < m < 50. A próxima linha contém o tempo tomado nas passadas de 100m que seu técnico registrou no cronômetro. O final de arquivo é marcado com m = 0, sendo que esta entrada não deve ser processada. O formato da entrada é dado por:

3
01min05s 02min15s 03min25s
0

Assim, ela nadou um total de 300m, sendo que o tempo necessário para cada passada de 100m foi de: 1min05s, 1min10s e 1min10s, respectivamente, nos primeiros 100m, na passagem dos 200m e nos últimos 100m. O valor min refere-se a minutos, e s aos segundos, ambos no intervalo [00; 59]. Esta sessão de treino sempre dura menos do que 1 hora.

Saída

Para cada caso de teste, o programa deverá imprimir em uma linha o melhor e o pior tempo entre as passagens de 100m, no formato abaixo:

01min05s 01min10s

Como curiosidade, isto é algo normal para um nadador olímpico, pois ele conta o número de braçadas e sabe com uma certa precisão o seu tempo a cada 100m.

Exemplo:

Entrada	Saída
3 01min05s 02min15s 03min25s 4 01min05s 02min04s 03min17s 04min25s 0	01min05s 01min10s 00min59s 01min13s

Problema H
Balas de Goma Esféricas

Nome do arquivo fonte: **gomas.c, gomas.cpp ou gomas.java**

Glória comprou um saco de balas de goma. No entanto, ela não quer carregá-las num saco plástico. Em vez disso, ela as coloca num tubo cilíndrico de diâmetro d. Supondo que cada uma das balas é uma esfera perfeita de raios R1, R2, ..., Rn, determine o menor comprimento do tubo que Glória deve usar para guardar suas balas (Figura 4).

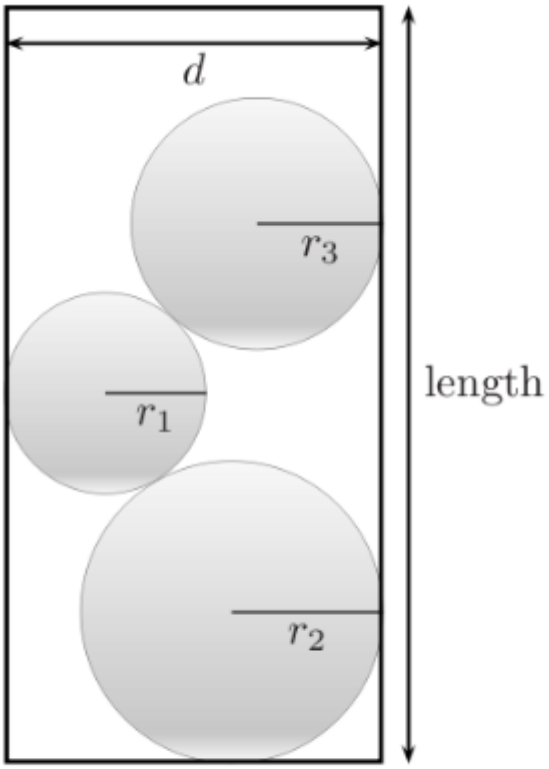


Figura 4: Tubo com balas de goma

Você deve supor que os raios das balas são suficientemente grandes para nunca três balas possam estar em contato simultaneamente uma com a outra dentro do tubo. Dada tal restrição, é útil levar em conta o fato de que as balas ficarão dispostas no tubo de tal modo que seus centros devem ficar sobre um plano 2-dimensional contendo o eixo de rotação do tubo.

Entrada

O arquivo de entrada consiste de múltiplos casos-teste. Cada caso-teste consiste de duas linhas. A primeira linha contém um inteiro n ($1 \leq n \leq 15$),

indicando o número de balas que Glória tem, e um valor em ponto flutuante d ($2 \leq d \leq 1000.0$) indicando o diâmetro do tubo cilíndrico, separados por um espaço. A segunda linha contém uma sequência de n pontos (separados por espaços) R1, R2, ... , Rn ($1.0 \leq Ri \leq d/2$), que denotam os raios das balas de goma. Uma linha em branco separa os casos-teste de entrada. Uma única linha com números "0 0" marca o fim da entrada; não processe este caso.

Saída

Para cada caso-teste de entrada, imprima o comprimento do menor tubo, arredondado para o inteiro mais próximo.

Exemplo:

Entrada	Saída
2 98.1789 42.8602 28.7622 3 747.702 339.687 191.953 330.811 0 0	138 1628

Problema I

Números Primos

Nome do arquivo fonte: **primos.c, primos.cpp ou primos.java**

Autor: João Víctor Rocon Maia

Hans deseja gerar alguns números primos para o seu sistema de criptografia. Vamos ajudar ele! Sua tarefa é gerar todos os números primos entre um intervalo numérico dado.

Entrada

A entrada começa com o número de casos de testes "t" em uma única linha ($t \leq 10$). Cada uma das "t" linhas seguintes vão conter dois números "m" e "n" ($1 \leq m \leq n \leq 10^9$, $(n-m) \leq 10^5$) separados por um espaço.

Saída

Para cada caso de teste imprima todos os números primos "p" de tal modo que $m \leq p \leq n$ (Ou seja, todos os números primos que estejam no intervalo [m,n]), um número por linha, de forma que os casos de teste são separados por uma linha em branco.

Exemplo:

Entrada	Saída
2 1 10 3 5	2 3 5 7 3 5

Problema J
Palitos de Fósforo

Nome do arquivo fonte: **fosforo.c, fosforo.cpp ou fosforo.java**

Um conjunto de palitos de fósforo é a ferramenta ideal para representar números. Uma forma comum de se representar os dez dígitos decimais usando palitos de fósforo é mostrada aqui em seguida:

Isto é feito de maneira idêntica nos despertadores digitais para mostrar os números. Com um dado número de palitos de fósforos, você pode gerar uma grande faixa de números. Nós estamos imaginando qual seria o menor e qual seria o maior número que você poderia criar usando todos os seus palitos de fósforo.

Entrada

Na primeira linha, um número inteiro positivo: o número de casos de teste, no máximo 100. Depois disto por caso de teste:

- Uma linha contendo um inteiro "n" ($2 \leq n \leq 100$): o número de palitos que você possui.

Saída

Por caso de teste:

- Uma linha contendo o maior e o menor número que você pode criar, separados por um único espaço. Ambos os números devem ser positivos e sem zero à esquerda.

Exemplo:

Entrada	Saída
4 3 6 7 15	7 7 6 111 8 711 108 7111111