CADERNO DE PROBLEMAS

18/05/2017 – 14h às 17h30

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 6 problemas, sendo que 2 deles estão descritos em inglês.

1. Não é permitido o uso de material impresso, livros, apostilas e dicionários. Apenas é permitido o uso de lápis, caneta, lapiseira, borracha, régua e papel para rascunho (o papel é fornecido pela comissão organizadora). O acesso à internet é bloqueado durante a realização da prova. A ajuda do ambiente de desenvolvimento como C, C++ ou Java pode ser utilizada.
2. A correção das resoluções dos problemas será automatizada, por meio do sistema de submissão eletrônica BOCA, tendo como base os resultados obtidos a partir de uma série de execuções dos algoritmos submetidos pelas equipes.
3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e da saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Os problemas não estão ordenados por ordem de dificuldade neste caderno. A sugestão é de que procure resolver primeiro os problemas mais fáceis.
5. Utilize os nomes indicados nos problemas para nomear os seus arquivos-fonte, de acordo com a linguagem de programação utilizada.
6. Os problemas devem ser resolvidos utilizando o raciocínio entrada-processamento-saída, ou seja, não é necessário armazenar toda a saída para depois exibi-la.

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (tela). Utilize as funções padrão para entrada e saída de dados:

- em C: `scanf`, `getchar`, `printf`, `putchar`;
- em C++: as mesmas de C ou os objetos `cin` e `cout`;
- em Java:
```
Scanner sc = new Scanner(); int i = sc.nextInt();
String s = sc.next(); float f = sc.nextFloat();
System.out.println(); System.out.printf("%d",i);
```

Em C ou C++: use `sprintf(str,"%d",num);` em vez de `itoa(num,str,10);`

Em Java: não utilize `package`.

Corte Retangular **(vermelha)**

Problema
A

Arquivo fonte: *corte.c, corte.cpp* ou *corte.java*

Na pequena aldeia histórica de Fippilônia, há uma atividade popular em cerimônias de casamento chamado corte retangular. Durante esta atividade, cada parente próximo da noiva vem e corta um retângulo no bolo de casamento (mas não pega um pedaço). O bolo tem uma forma retangular. O problema é contar quantos pedaços estão no bolo após os cortes retangulares.

Por exemplo, na Figura 1, o tamanho do bolo é 3 × 5, e três pessoas fizeram cortes retangulares no bolo.

Como resultado, o bolo é cortado em seis pedaços.

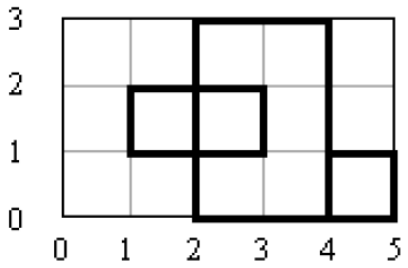


Figura 1: Três cortes retangulares em um bolo 3 × 5.

Cada corte retangular é especificado pelas coordenadas (*x, y*) de seus dois cantos opostos. A entrada para a figura acima pode ser encontrada na primeira amostra de entrada. Como há famílias numerosas em Fippilônia, o número de pedaços pode ser grande e precisamos de um programa de computador para calculá-lo.

Entrada

A entrada contém vários casos de teste. Cada teste tem várias linhas. A primeira linha tem dois inteiros w ($1 \leq w \leq 20$) e h ($1 \leq h \leq 20$), que são a largura e a altura do bolo. A segunda linha contém um único inteiro n ($0 \leq n \leq 50$), que é o número de pessoas que cortam retângulos no bolo. Depois disto, existem n linhas cada uma contendo os inteiros x_1, y_1, x_2, y_2, que são as coordenadas de dois cantos opostos do corte. Você pode assumir $0 \leq x_1, x_2 \leq w$; $x_1 \neq x_2$; $0 \leq y_1, y_2 \leq h$; e $y_1 \neq y_2$. A última linha da entrada é uma linha contendo dois zeros.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, escreva o número de pedaços em que o bolo é cortado.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
5 3 3 1 1 3 2 4 0 2 3 4 0 5 1 6 6 2 2 0 5 3 3 1 4 2 0 0	6 3

Árvore (amarela)

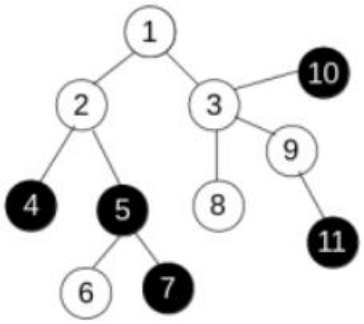
Problema B

Arquivo fonte: *arvore.c*, *arvore.cpp* ou *arvore.java*

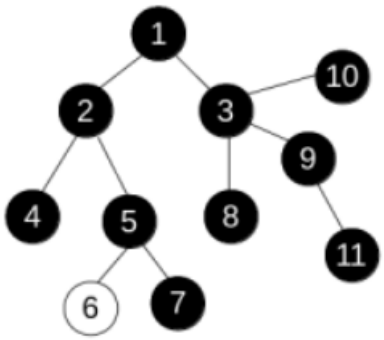
Anton está cultivando uma árvore em seu jardim. No caso de você ter esquecido, uma árvore é um grafo não direcionado acíclico conectado.

Existem *n* vértices na árvore, cada um deles é pintado de preto ou branco. Anton não gosta de árvores multicoloridas, então ele quer mudar a árvore de modo que todos os vértices tenham a mesma cor (preto ou branco).

Para alterar as cores Anton pode usar apenas operações de um tipo. Ele denotou como pintura (*v*), onde *v* é algum vértice da árvore. Esta operação altera a cor de todos os vértices *u* de modo que todos os vértices no caminho mais curto de *v* para *u* tenham a mesma cor (incluindo *v* e *u*). Por exemplo, considere a árvore.



e aplicando a operação *pintura* (3) irá obter a seguinte árvore:



Anton está interessado no número mínimo de operações que ele precisa realizar para tornar as cores de todos os vértices iguais.

Entrada

A primeira linha da entrada contém um único inteiro n ($1 \le n \le 200.000$) – o número de vértices da árvore.

A segunda linha contém n inteiros cor_i ($0 \le cor_i \le 1$) – cores dos vértices. $cor_i = 0$ significa que o i-ésimo vértice é inicialmente pintado de branco, enquanto $cor_i = 1$ significa que ele é inicialmente pintado de preto.

Em seguida tem-se n-1 linhas, cada uma delas contendo um par de inteiros u_i e v_i ($1 \leq u_i, v_i \leq n, u_i \neq v_i$) – índices de vértices conectados pela borda correspondente. É garantido que todos os pares (u_i, v_i) são distintos, isto é, não existem bordas múltiplas.

O final da entrada é indicado por $n = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, imprimir um inteiro – o número mínimo de operações que Anton deve aplicar para tornar todos os vértices da árvore pretos ou todos os vértices da árvore brancos.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
11 0 0 0 1 1 0 1 0 0 1 1 1 2 1 3 2 4 2 5 5 6 5 7 3 8 3 9 3 10 9 11 4 0 0 0 0 1 2 2 3 3 4 0	2 0

Mr. Bean (azul)

Problema C

Source file: *bean.c*, *bean.cpp* or *bean.java*

Mr. Bean used to have a lot of problems packing his suitcase for holiday. So he is very careful for this coming holiday. He is more serious this time because he is going to meet his fiancée and he is also keeping frequent communication with you as a programmer friend to have suggestions. He gets confused when he buys a gift box for his fiancée because he can't decide whether it will fit in his suitcase or not. Sometimes a box doesn't fit in his suitcase in one orientation and after rotating the box to a different orientation it fits in the suitcase. This type of behavior makes him puzzled.

So to make things much simpler he bought another suitcase having same length, width and height, which is 20 inches. This measurement is taken from inside of the box. So a box which has length, width and height of 20 inches will just t in this suitcase. He also decided to buy only rectangular shaped boxes and keep a measuring tape in his pocket. Whenever he chooses one gift box, which must be rectangular shaped, he quickly measures the length, width and height of the box. But still he can't decide whether it will fit in his suitcase or not. Now he needs your help. Please write a program for him which calculates whether a rectangular box fits in his suitcase or not provided the length, width and height of the box. Note that, sides of the box must be parallel to the sides of the suitcase.

Input

Input starts with an integer T ($T \leq 100$), which indicates the number of test cases.

Each of the next T line contains three integers L, W and H ($1 \leq L, W, H \leq 50$) denoting the length, width and height of a rectangular shaped box.

The input must be read from standard input.

Output

For each test case, output a single line. If the box fits in the suitcase in any orientation having the sides of the box is parallel to the sides of the suitcase, this line will be **'Case #: good'**, otherwise it will be **'Case #: bad'**. In your output # will be replaced by the case number.

Please see the sample input and sample output for exact format.

The output must be written to standard output.

Sample Input

```
2
20 20 20
1 2 21
```

Output for the sample Input

```
Case 1: good
Case 2: bad
```

Wavelet Compression (verde)

Problema D

Source file: *wavelet.c*, *wavelet.cpp* or *wavelet.java*

Description

The discrete wavelet transform is a popular tool for signal compression. In this problem, your job is to write a program to decompress a one-dimensional signal (a list of integers) that has been compressed by a simple wavelet transform.

To understand how this simple wavelet transform works, suppose that we have a list of an even number of integers. We compute the sum and difference of each pair of consecutive samples, resulting in two lists of sums and differences each having half the original length. Formally, if the original samples are

$$a(1),\ldots, a(n)$$

the i-th sum $s(i)$ and difference $d(i)$ are computed as:

$$\text{for } i = 1,\ldots, n/2 :$$

$$s(i) = a(2 \times i - 1) + a(2 \times i)$$

$$d(i) = a(2 \times i - 1) - a(2 \times i)$$

This is then rearranged to give the transformed signal by first listing the sums and then the differences.

For example, if the input signal is:

$$5, 2, 3, 2, 5, 7, 9, 6$$

Then the sum and difference signals are:

$$s(i) = 7, 5, 12, 15$$

$$d(i) = 3, 1, -2, 3$$

Thus, the transformed signal is:

$$7, 5, 12, 15, 3, 1, -2, 3$$

The same process is applied recursively to the first half of the transformed signal, treating $s(i)$ as the input signal, until the length of the input signal is 1. In the example above, the final transformed signal is:

$$39, -15, 2, -3, 3, 1, -2, 3$$

It is assumed that the length of the original input is a power of 2, and the input signal consists of integers between 0 and 255 (inclusive) only.

Input

The input consists of a number of cases. Each case is specified on a line, starting with an integer N ($1 \le N \le 256$) indicating the number of samples. The next N integers are the transformed samples.

The end of input is indicated by a case in which $N = 0$.

The input must be read from standard input.

Output

For each test case, output the original samples on a single line, separated by a single space.

The output must be written to standard output.

Sample input	Output for the sample input
8 39 -15 2 -3 3 1 -2 3 4 10 -4 -1 -1 0	5 2 3 2 5 7 9 6 1 2 3 4

Rota de ônibus (rosa)

Problema E

Arquivo fonte: *rota.c*, *rota.cpp* ou *rota.java*

Existem n cidades situadas ao longo da estrada principal de Fipplândia. As cidades são representadas por suas coordenadas – números inteiros $a_1, a_2, ..., a_n$. Todas as coordenadas são distintas por pares.

É possível ir de uma cidade a outra apenas de ônibus. Mas todos os ônibus e estradas são muito velhos e o ministro dos transportes decidiu construir uma nova rota de ônibus. O ministro não quer gastar grandes quantias de dinheiro – ele quer escolher duas cidades de tal maneira que a distância entre elas é a mínima possível. A distância entre duas cidades é igual ao valor absoluto da diferença entre suas coordenadas.

É possível que haja múltiplos pares de cidades com distância mínima possível, de modo que o ministro quer saber a quantidade de tais pares.

Sua tarefa é escrever um programa que irá calcular a distância mínima possível entre dois pares de cidades e a quantidade de pares que têm essa distância.

Entrada

A primeira linha contém um número inteiro n ($2 \leq n \leq 2 \cdot 10^5$).

A segunda linha contém n números inteiros $a_1, a_2, ..., a_n$ ($-10^9 \leq a_1 \leq 10^9$). Todos os números a_1 são pares distintos.

O final da entrada é indicado por $n = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Imprimir dois números inteiros – a distância mínima e a quantidade de pares com essa distância.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
4 6 -3 0 4 3 -2 0 2 0	2 1 2 2

Fractal (branca)

Problema F

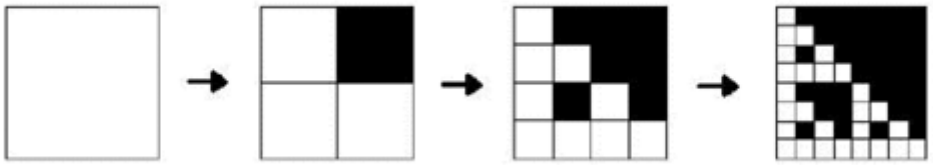
Arquivo fonte: *fractal.c*, *fractal.cpp* ou *fractal.java*

Desde que Kotozicks, um famoso abstracionista de Fipplândia, ouviu falar de fractais, ele fez deles o tema principal de suas telas. Todas as manhãs, o artista pega um pedaço de papel milimetrado e começa a fazer um modelo de sua futura tela. Ele pega um quadrado tão grande quanto $n \times n$ quadrados e pinta alguns deles em preto. Então ele pega um pedaço de papel quadrado limpo e pinta o fractal usando o seguinte algoritmo:

Passo 1. O papel é dividido em n^2 quadrados idênticos e alguns deles são pintados de preto de acordo com o modelo.

Passo 2. Cada quadrado que permanece branco é dividido em n^2 quadrados menores e alguns deles são pintados de preto de acordo com o modelo.

Cada passo seguinte repete o passo 2.



Infelizmente, este trabalho cansativo exige muito tempo do gênio da pintura. Kotozicks tem sonhado de fazer o processo automático para passar a fazer fractais 3D ou mesmo 4D.

Entrada

A primeira linha contém números inteiros n e k ($2 \leq n \leq 3$, $1 \leq k \leq 5$), sendo k a quantidade de passos do algoritmo. Cada uma das seguintes n linhas contém n símbolos que determinam o modelo. O símbolo "." significa um quadrado branco, enquanto "*" representa um quadrado preto. É garantido que o modelo tenha pelo menos um quadrado branco.

O final da entrada é indicado por $n = 0$ e $k = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Exibir na saída uma matriz $n^k \times n^k$ que é o que uma imagem deve ser semelhante após k passos do algoritmo.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	Saída para o exemplo de entrada
2 3 .* .. 3 2 .*. *** .*. 0 0	.******* ..****** .*.***** ....**** .***.*** ..**..** .*.*.*.* . .*.***.*. ********* .*.***.*. ********* ********* ********* .*.***.*. ********* .*.***.*.