3^a Seletiva da UFPR

Maratona de Programação dos Alunos da UFPR – 2012

Sevidor BOCA:

`http://maratona.c3sl.ufpr.br/boca/`

Organizadores:

André Luiz Guedes, Vinicius Kwiecien Ruoso, Ricardo Tavares de Oliveira

Lembretes:

- É permitido consultar livros, anotações ou qualquer outro material impresso durante a prova.
- A correção é automatizada, portanto, siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa. Deve-se considerar entradas e saídas padrão.
- Procure resolver o problema de maneira eficiente. Se o tempo superar o limite pré-definido, a solução não é aceita. As soluções são testadas com outras entradas além das apresentadas como exemplo dos problemas.
- Teste seu programa antes de submetê-lo. A cada problema detectado (erro de compilação, erro em tempo de execução, solução incorreta, formatação imprecisa, tempo excedido ...), há penalização de 20 minutos. O tempo é critério de desempate entre duas ou mais equipes com a mesma quantidade de problemas resolvidos.
- Utilize o *clarification* para dúvidas da prova. Os juízes podem opcionalmente atendê-lo com respostas acessíveis a todos.

3^a Seletiva da UFPR

Maratona de Programação dos Alunos da UFPR

28 de julho de 2011

Sumário

Problema A: Ants Colony

`File: ants.[c|cpp|java|pas]`

A group of ants is really proud because they built a magnificent and large colony. However, the enormous size has become a problem, because many ants do not know the path between some parts of the colony. They desperately need your help!

The colony of ants was made as a series of N anthills, connected by tunnels. The ants, obsessive as they are, numbered the anthills sequentially as they built them. The first anthill, numbered 0, did not require any tunnel, but for each of the subsequent anthills, 1 through $N-1$, the ants also built a single tunnel that connected the new anthill to one of the existing anthills. Of course, this tunnel was enough to allow any ant to go to any previously built anthill, possibly passing through other anthills in its path, so they did not bother making extra tunnels and continued building more anthills.

Your job is, given the structure of the colony and a set of queries, calculate for each query the shortest path between pairs of anthills. The length of a path is the sum of the lengths of all tunnels that need to be traveled.

Input

Each test case is given using several lines. The first line contains an integer N representing the number of anthills in the colony ($2 \leq N \leq 10^5$). Each of the next $N-1$ lines contains two integers that describe a tunnel. Line i, for $1 \leq i \leq N-1$, contains A_i and L_i , indicating that anthill i was connected directly to anthill A_i with a tunnel of length L_i ($0 \leq A_i \leq i-1$ and $1 \leq L_i \leq 10^9$). The next line contains an integer Q representing the number of queries that follow ($1 \leq Q \leq 10^5$). Each of the next Q lines describes a query and contains two distinct integers S and T ($0 \leq S, T \leq N-1$), representing respectively the source and target anthills. The last test case is followed by a line containing one zero.

Output

For each test case output a single line with Q integers, each of them being the length of a shortest path between the pair of anthills of a query. Write the results for each query in the same order that the queries were given in the input.

Sample input	**Sample output**
6 0 8 1 7 1 9 0 3 4 2 4 2 3 5 2 1 4 0 3 2 0 1 2 1 0 0 1 6 0 1000000000 1 1000000000 2 1000000000 3 1000000000 4 1000000000 1 5 0 0	16 20 11 17 1 1 5000000000

Problema B: Livro Caixa

`Arquivo: caixa.[c|cpp|java|pas]`

A FCC (Fundação de Combate à Corrupção) desmontou um grande esquema de corrupção na Nlogônia. Durante a operação, foram apreendidos diversos cadernos e livros com anotações documentando as transações ilícitas realizadas pelo esquema.

Vários desses livros contém páginas com os valores de várias transações em nilogos (a moeda local da Nlogônia, cujo símbolo é N$) e o fluxo de caixa resultante dessas transações. Por exemplo, se em uma página foi registrada uma entrada de N$ 7, uma entrada de N$ 2, uma saída de N$ 3, uma entrada de N$ 1 e outra saída de N$ 11, o fluxo de caixa nesta página é $7 + 2 - 3 + 1 - 11 = -4$.

No entanto, para dificultar o trabalho da polícia, os contraventores não anotaram em seus livros qual o tipo de cada transação. No exemplo acima, as anotações na página seriam apenas 7, 2, 3, 1 e 11 (sem indicação se elas são entradas ou saídas). O fluxo de caixa de cada página sempre é anotado normalmente, com o sinal (no caso, -4).

Para obter a condenação dos contraventores, os promotores precisam poder afirmar com certeza se cada operação foi uma entrada ou uma saída. No exemplo acima, a transação de N$ 7 certamente foi uma entrada, e a transação de N$ 11 certamente foi uma saída. Mas, não se pode afirmar nada sobre as transações de N$ 2, N$ 3, e N$ 1. As transações de N$ 2 e N$ 1 poderiam ter sido entradas e a transaçãoo de N$ 3 uma saída, ou N$ 2 e N$ 1 poderiam ter sido saídas e a transação de N$ 3 uma uma entrada.

Muitos cadernos possuem números relativamente grandes, com muitas transações, então é difícil para a polícia reconstruir o histório de operações. Por isso, eles precisam de um programa que o faça de forma eficiente.

Entrada

A entrada consiste de vários casos de teste. A primeira linha da entrada contém dois inteiros N e F, indicando respectivamente o número de operações na página ($2 \leq N \leq 40$) e o fluxo de caixa para esta página ($-16000 \leq F \leq 16000$). Cada uma das N linhas seguintes contém um inteiro T_i indicando o valor da i-ésima transação ($1 \leq T_i \leq 1000$).

O último caso de teste é seguido por uma linha que contém apenas dois zeros separados por espaços em branco.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha com N caracteres. O i-ésimo caractere deve ser '+', se for possível afirmar com certeza que a i-ésima operação foi uma entrada, '-', se for possível afirmar com certeza que a i-ésima operação foi uma saída, e '?', se não for possível determinar com certeza qual o tipo da operação.

Caso não exista uma sequência de entradas e saídas que totalize o fluxo de caixa indicado, imprima uma única linha contendo o caractere '*'.

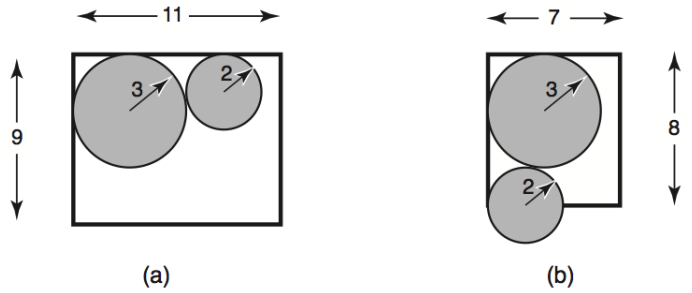
Exemplo de entrada	**Exemplo de saída**
5 7 1 2 3 4 5 4 15 3 5 7 11 5 12 6 7 7 1 7 0 0	?+??+ * +??-?

Problema C: Elevador

`Arquivo: elevador.[c|cpp|java|pas]`

A FCC (Fábrica de Cilindros de Carbono) fabrica de vários tipos de cilindros de carbono. A FCC está instalada no décimo andar de um prédio, e utiliza os vários elevadores do prédio para transportar os cilindros. Por questão de segurança, os cilindros devem ser transportados na posição vertical; como são pesados, no máximo dois cilindros podem ser transportados em uma única viagem de elevador. Os elevadores têm formato de paralelepípedo e sempre têm altura maior que a altura dos cilindros.

Para minimizar o número de viagens de elevador para transportar os cilindros, a FCC quer, sempre que possível, colocar dois cilindros no elevador. A figura abaixo ilustra, esquematicamente (vista superior), um caso em que é isto possível (a), e um caso em que isto não é possível (b):



Como existe uma quantidade muito grande de elevadores e de tipos de cilindros, a FCC quer que você escreva um programa que, dadas as dimensões do elevador e dos dois cilindros, determine se é possível colocar os dois cilindros no elevador.

Entrada

A entrada contém vários casos de teste. A primeira e única linha de cada caso de teste contém quatro números inteiros L, C, R_1 e R_2, separados por espaços em branco, indicando respectivamente a largura do elevador ($1 \leq L \leq 100$), o comprimento do elevador ($1 \leq C \leq 100$), e os raios dos cilindros ($1 \leq R_1, R_2 \leq 100$).

O último caso de teste é seguido por uma linha que contém quatro zeros separados por espaços em branco.

Saída

Para cada caso de teste, o seu programa deve imprimir uma única linha com um único caractere: 'S' se for possível colocar os dois cilindros no elevador e 'N' caso contrário.

Exemplo de entrada	**Exemplo de saída**

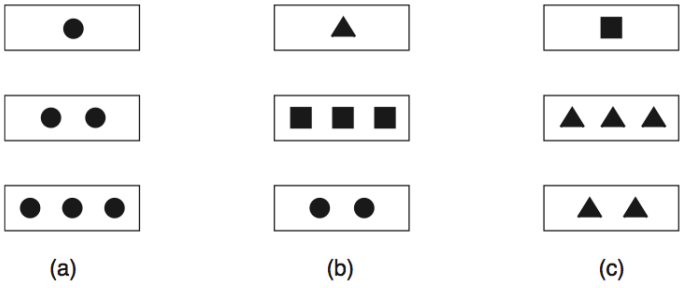
`11 9 2 3`
`7 8 3 2`
`10 15 3 7`
`8 9 3 2`
`0 0 0 0`
	`S`
	`N`
	`N`
	`S`

Problema D: Set

`Arquivo: set.[c|cpp|java|pas]`

Set é um jogo jogado com um baralho no qual cada carta pode ter uma, duas ou três figuras. Todas as figuras em uma carta são iguais, e podem ser círculos, quadrados ou triângulos.

Um set é um conjunto de três cartas em que, para cada característica (número e figura), ou as três cartas são iguais, ou as três cartas são diferentes. Por exemplo, na figura abaixo, (a) é um set válido, já que todas as cartas têm o mesmo tipo de figura e todas elas têm números diferentes de figuras. Em (b), tanto as figuras quanto os números são diferentes para cada carta. Por outro lado, (c) não é um set, já que as duas últimas cartas têm a mesma figura, mas esta é diferente da figura da primeira carta.



O objetivo do jogo é formar o maior número de sets com as cartas que estão na mesa; cada vez que um set é formado, as três cartas correspondentes são removidas de jogo.

Quando há poucas cartas na mesa, é fácil determinar o maior número de sets que podem ser formados; no entanto, quando há muitas cartas há muitas combinações possíveis. Seu colega quer treinar para o campeonato mundial de Set, e por isso pediu que você fizesse um programa que calcula o maior número de sets que podem ser formados com um determinado conjunto de cartas.

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro N ($3 \leq N \leq 3x104$), indicando o número de cartas na mesa; cada uma das N linhas seguintes contém a descrição de uma carta.

A descrição de uma carta é dada por duas palavras separadas por um espaço; a primeira, "um", "dois" ou "tres", indica quantas figuras aquela carta possui. A segunda, "circulo" (ou "circulos"), "quadrado" (ou "quadrados") ou "triangulo" (ou "triangulos") indica qual tipo de figura está naquela carta. O final da entrada é indicado por uma linha contendo um zero.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo um número inteiro, indicando o maior número de sets que podem ser formados com as cartas dadas.

Exemplo de entrada	**Exemplo de saída**
3 um circulo dois circulos tres circulos 3 um triangulo tres quadrados dois circulos 6 dois quadrados dois quadrados dois quadrados dois quadrados dois quadrados dois quadrados 4 um quadrado tres triangulos um quadrado dois triangulos 0	1 1 2 0

Problema E: Hooligan

File: `hooligan.[c|cpp|java|pas]`

Soccer is the American English word used to describe Football, the British English word applied to the most popular sport in Latin America (and in the world). Hooligan is sometimes used to describe an aggressive, troublemaking, soccer fan.

In Linearonia, a soccer tournament is in progress. There, the ranking process is as follows: for each game, the winner gets two points and the loser gets no points; in case of a tie, both competitors get one point each. The champion is the team with the highest number of points. Every pair of distinct teams play against each other exactly the same number of times, called the matching number.

You have your favorite team, your dream team, and you wonder whether it is possible for your dream team to be champion. You know the number of teams, the matching number and the results of some games that have already been played. Write a program to decide whether at the end of the tournament your dream team can still be the only champion, with strictly more points than any other team.

Input

The input contains several test cases, each case consists of one or more lines. The first line contains three integers, N, M and G, separated by single spaces, representing respectively the number of teams playing in the tournament ($2 \leq N \leq 40$), the matching number ($1 \leq M \leq 4$) and the number of games already played ($1 \leq G$). Your dream team is identified by the number 0, the other teams are identified by the integers $1, 2, ..., N-1$.

Each of the next G lines describes a game already played. The line contains an integer I, a character C and an integer J, separated by single spaces. Integers I and J are the teams that played that game ($I \neq J$ and $0 \leq I, J \leq N-1$). Character C is '$<$' if team I lost to team J, or '$=$' if the game ended in a tie.

The last test case is followed by a line containing three zeros separated by single spaces.

Output

For each test case in the input, your program must print a single line, containing one single character, uppercase 'Y' if your dream team can be the champion, or uppercase 'N' otherwise.

Sample input	Sample output
4 2 6 0 < 3 3 = 2 2 < 0 1 < 0 2 = 0 3 < 0 4 1 5 2 = 0 0 < 1 1 = 3 2 < 1 0 < 3 4 2 5 2 = 0 0 < 1 1 = 3 2 < 1 0 < 3 2 1 1 1 < 0 4 1 1 0 < 1 4 1 2 0 < 1 0 < 2 0 0 0	Y N Y Y Y N

Problema F: Jingle Composing

File: `jingle.[c|cpp|java|pas]`

A. C. Marcos is taking his first steps in the direction of jingle composition. He is having some troubles, but at least he is achieving pleasant melodies and attractive rhythms.

In music, a note has a pitch (its frequency, resulting in how high or low is the sound) and a duration (for how long the note should sound). In this problem we are interested only in the duration of the notes.

A jingle is divided into a sequence of measures, and a measure is formed by a series of notes. The duration of a note is indicated by its shape. In this problem, we will use uppercase letters to indicate a note's duration. The following table lists all the available notes:

Notes	W	H	Q	E	S	T	X
Duration	1	1/2	1/4	1/8	1/16	1/32	1/64

The duration of a measure is the sum of the durations of its notes. In Marcos' jingles, each measure has the same duration. As Marcos is just a beginner, his famous teacher Johann Sebastian III taught him that the duration of a measure must always be 1.

For example, Marcos wrote a composition containing five measures, of which the first four have the correct duration and the last one is wrong. In the example below, each measure is surrounded with slashes and each note is represented as in the table above.

$$/HH/QQQQ/XXXTXTEQH/W/HW/$$

Marcos likes computers as much as music. He wants you to write a program that determines, for each one of his compositions, how many measures have the right duration.

Input

The input contains several test cases. Each test case is described in a single line containing a string whose length is between 3 and 200 characters, inclusive, representing a composition. A composition begins and ends with a slash '/'. Measures in a composition are separated by a slash '/'. Each note in a measure is represented by the corresponding uppercase letter, as described above. You may assume that each composition contains at least one measure and that each measure contains at least one note. All characters in the input will be either slashes or one of the seven uppercase letters used to represent notes, as described above.

The last test case is followed by a line containing a single asterisk.

Output

For each test case your program must output a single line, containing a single integer, the number of measures that have the right duration.

Sample input	**Sample output**
/HH/QQQQ/XXXTXTEQH/W/HW/ /W/W/SQHES/ /WE/TEX/THES/ *	4 3 0

Problema G: Brothers

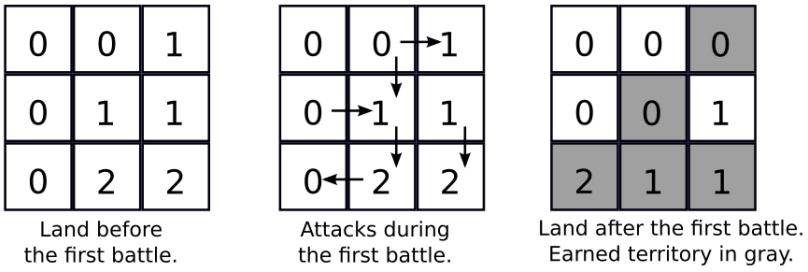
File: brothers.[c|cpp|java|pas]

In the land of ACM ruled a great King who became obsessed with order. The kingdom had a rectangular form, and the King divided the territory into a grid of small rectangular counties. Before dying, the King distributed the counties among his sons.

However, he was unaware that his children had developed a strange rivalry: the first heir hated the second heir, but not the rest; the second heir hated the third heir, but not the rest, and so on ... Finally, the last heir hated the first heir, but not the other heirs.

As soon as the King died, the strange rivalry among the King's sons sparked off a generalized war in the kingdom. Attacks only took place between pairs of adjacent counties (adjacent counties are those that share one vertical or horizontal border). A county X attacked an adjacent county Y whenever the owner of X hated the owner of Y . The attacked county was always conquered by the attacking brother. By a rule of honor all the attacks were carried out simultaneously, and a set of simultaneous attacks was called a battle. After a certain number of battles, the surviving sons made a truce and never battled again.

For example, if the King had three sons, named 0, 1 and 2, the figure below shows what happens in the first battle for a given initial land distribution:



You were hired to help an ACM historian determining, given the number of heirs, the initial land distribution and the number of battles, what was the land distribution after all battles.

Input

The input contains several test cases. The first line of a test case contains four integers N, R, C and K, separated by single spaces. N is the number of heirs ($2 \leq N \leq 100$), R and C are the dimensions of the kingdom ($2 \leq R, C \leq 100$), and K is the number of battles ($1 \leq K \leq 100$). Heirs are identified by sequential integers starting from zero (0 is the first heir, 1 is the second heir, ..., $N - 1$ is the last heir). Each of the next R lines contains C integers $H_{r,c}$ separated by single spaces, representing initial land distribution: $H_{r,c}$ is the initial owner of the county in row r and column c ($0 \leq H_{r,c} \leq N - 1$).

The last test case is followed by a line containing four zeroes separated by single spaces.

Output

For each test case, your program must print R lines with C integers each, separated by single spaces in the same format as the input, representing the land distribution after all battles.

Sample input	**Sample output**
3 4 4 3 0 1 2 0 1 0 2 0 0 1 2 0 0 1 2 2 4 2 3 4 1 0 3 2 1 2 8 4 2 1 0 7 1 6 2 5 3 4 0 0 0 0	2 2 2 0 2 1 0 1 2 2 2 0 0 2 0 0 1 0 3 2 1 2 7 6 0 5 1 4 2 3

Problema H: Lazy Jumping Frog

`File: frog.[c|cpp|java|pas]`

Mr. Frog lives in a grid-like marsh of rectangular shape, composed of equally-sized cells, some of which are dry, some of which are only watery places. Mr. Frog lives in a dry cell and can jump only from a dry cell to another dry cell on his wanderings around the marsh.

Mr. Frog wants to visit his girlfriend, Ms. Toad, who also lives in a dry cell in the same marsh. But Mr. Frog is lazy, and wants to spend the minimum amount of energy in his jumping way to Ms. Toad's home. Mr. Frog knows how much energy he spends in any of his jumps. For any single jump, Mr. Frog always uses the following figure to determine which are the possible target cells from his current position (the cell marked F), and the corresponding energy spent in the jump, in calories. Any other cell is unreachable from Mr. Frog's current position with a single jump.

7	6	5	6	7
6	3	2	3	6
5	2	**F**	2	5
6	3	2	3	6
7	6	5	6	7

Your task is to determine the minimum amount of energy that Mr. Frog needs to spend to get from his home to Ms. Toad's home.

Input

The input contains several test cases. The first line of a test case contains two integers, C and R, indicating respectively the number of columns and rows of the marsh ($1 \leq C$, $R \leq 1000$). The second line of a test case contains four integers C_f, R_f, C_t, and R_t, where (C_f,R_f) specify Mr. Frog's home location and (C_t,R_t) specify Ms. Toad's home location ($1 \leq C_f$, $C_t \leq C$ and $1 \leq R_f$, $R_t \leq R$). The third line of a test case contains an integer W ($0 \leq W \leq 1000$) indicating the number of watery places in the marsh. Each of the next W lines contains four integers C_1, R_1, C_2, and R_2 ($1 \leq C_1 \leq C_2 \leq C$ and $1 \leq R_1 \leq R_2 \leq R$) describing a rectangular watery place comprising cells whose coordinates (x, y) are so that $C_1 \leq x \leq C_2$ and $R_1 \leq y \leq R_2$. The end of input is indicated by $C = R = 0$.

Output

For each test case in the input, your program must produce one line of output, containing the minimum calories consumed by Mr. Frog to go from his home location to Ms. Toad's home location. If there is no way Mr. Frog can get to Ms. Toad's home, your program should output impossible.

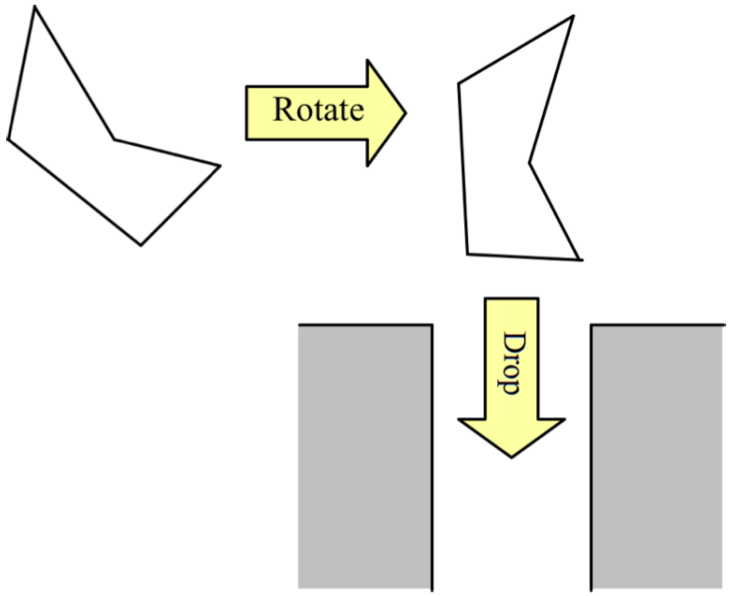
Sample input	Sample output
4 4 1 1 4 2 2 2 1 3 3 4 3 4 4 4 4 1 1 4 2 1 2 1 3 4 7 6 4 2 7 6 5 4 1 7 1 5 1 5 5 2 4 3 4 7 5 7 5 6 6 6 6 0 0	14 impossible 12

Problema I: Trash Removal

`File: trash.[c|cpp|java|pas]`

Allied Chute Manufacturers is a company that builds trash chutes. A trash chute is a hollow tube installed in buildings so that trash dropped in at the top will fall down and be collected in the basement. Designing trash chutes is actually highly nontrivial. Depending on what kind of trash people are expected to drop into them, the trash chute needs to have an appropriate size. And since the cost of manufacturing a trash chute is proportional to its size, the company always would like to build a chute that is as small as possible. Choosing the right size can be tough though.

We will consider a 2-dimensional simplification of the chute design problem. A trash chute points straight down and has a constant width. Objects that will be dropped into the trash chute are modeled as polygons. Before an object is dropped into the chute it can be rotated so as to provide an optimal fit. Once dropped, it will travel on a straight path downwards and will not rotate in flight. The following figure shows how an object is first rotated so it fits into the trash chute.



Your task is to compute the smallest chute width that will allow a given polygon to pass through.

Input

The input contains several test cases. Each test case starts with a line containing an integer N ($3 \leq N \leq 100$), the number of points in the polygon that models the trash item.

The next N lines then contain pairs of integers x_i and y_i ($0 \leq x_i, y_i \leq 104$), giving the coordinates of the polygon vertices in order. All points in one test case are guaranteed to be mutually distinct and the polygon sides will never intersect. (Technically, there is one inevitable exception of two neighboring sides sharing their common vertex. Of course, this is not considered an intersection.)

The last test case is followed by a line containing a single zero.

Output

For each test case, display its case number followed by the width of the smallest trash chute through which it can be dropped. Display the minimum width with exactly two digits to the right of the decimal point, rounding up to the nearest multiple of 1/100. Answers within 1/100 of the correct rounded answer will be accepted.

Follow the format of the sample output.

Sample input	Sample output
3 0 0 3 0 0 4 4 0 10 10 0 20 10 10 20 0	Case 1: 2.40 Case 2: 14.15

Problema J: Balé

`Arquivo: bale.[c|cpp|java|pas]`

Uma academia de balé irá organizar uma Oficina de Balé Intensivo (OBI) na Semana de Balé Contemporâneo (SBC). Nessa academia, existem N bailarinas que praticam regularmente. O dono da academia, por ser experiente, consegue medir o nível de habilidade de cada uma delas por um número inteiro; nessa medição, números maiores correspondem a dançarinas mais habilidosas, e os números obtidos são todos distintos. Além disso, ele possui uma lista das bailarinas em ordem cronológica de ingresso na academia: As bailarinas que aparecem primeiro na lista estão há mais tempo na academia, e as que estão no final ingressaram mais recentemente.

O dono da academia decidiu escolher duas das bailarinas para ajudá-lo na realização do evento: uma ajudará no trabalho braçal, enquanto a outra irá exemplificar os passos de balé. Por seu perfeccionismo, ele deseja que a bailarina que exemplificará os passos de dança seja, dentre as duas meninas do par, a mais habilidosa e a que frequenta a academia há mais tempo.

Ele sabe que a Oficina será um sucesso desde que os dois critérios mencionados acima sejam satisfeitos pela dupla de dançarinas escolhidas. Com isso, ele ficou curioso para saber quantas duplas de dançarinas podem ajudá-lo na Oficina. A quantidade de dançarinas, contudo, é relativamente grande e ele não possui nem tempo nem paciência para fazer tal cálculo. Como vocês são amigos, ele pediu a sua ajuda para contar quantas duplas são válidas. Você pode ajudá-lo?

Por exemplo, digamos que a academia possua 5 dançarinas com níveis de habilidade 1, 5, 2, 4 e 3, onde a primeira, que possui nível '1', está na academia há mais tempo e a última, que possui nível '3', está há menos. Temos, então, 4 possíveis duplas que poderemos usar nesta Oficina, que são (5,2), (5,4), (5,3) e (4,3). Note que a dupla (1,3), por exemplo, não pode ser escolhida pelo dono da academia, pois a mais habilidosa dentre as duas é também a mais nova da dupla.

Entrada

A primeira linha contém um número N ($1 \leq N \leq 100000$), que representa a quantidade de dançarinas que estão registradas na academia. A segunda linha da entrada contém N inteiros a_i ($1 \leq a_i \leq 100000$), onde o primeiro inteiro é o nível da dançarina que está há mais tempo na academia, o segundo inteiro é o nível da próxima dançarina mais antiga na academia (mas mais nova que a dançarina anterior), e assim sucessivamente. O total de pares válidos, em todos os casos, será ≤ 1000000.

O final da entrada é indicado por uma linha contendo um zero.

Saída

Para cada caso de teste da entrada seu programa deve imprimir uma única linha na saída, contendo um número inteiro, indicando o total de duplas de dançarinas válidas para essa Oficina, dadas as regras descritas anteriormente.

Sample input	**Sample output**
5 1 5 2 4 3 9 9 8 7 6 5 4 3 1 2 0	4 35