Problem A. Approximation

Input file:	`approximation.in`
Output file:	`approximation.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

Flatland spy Zilrits is on mission in Edgeland. Recently he had to transmit some very important secret message to the base using his nanotransmitter. Unfortunately, nanoreceiver at the base was out of order, so the received message was corrupted.

The base officers know that the message that Zirlits had to transmit was a non-decreasing sequence of real numbers. However, they received some arbitrary sequence of integers $a_1, a_2, \ldots, a_n$. Now they decided to try to restore the most probable original sequence. To do this they would like to find non-decreasing sequence $b_1, b_2, \ldots, b_n$ of real numbers such that the sum

$$s = \sum_{i=1}^{n} (a_i - b_i)^2$$

is as small as possible.

Help them to find such sequence.

Input

The first line of the input file contains n — the length of the received sequence. The second line contains n integer numbers: $a_1, a_2, \ldots, a_n$ ($1 \le n \le 200\,000$, $1 \le a_i \le 10^6$).

Output

Output n numbers: the most probable original sequence. If there are several solutions, output any one. For your answer s must have absolute or relative error not exceeding 10^{-9}.

Examples

approximation.in	approximation.out
5 5 4 3 2 1	3 3 3 3 3
9 3 2 1 8 6 4 9 7 5	2.0 2.0 2.0 6.0 6.0 6.0 7.0 7.0 7.0

Problem B. Big Set

Input file:	`bigset.in`
Output file:	`bigset.out`
Time limit:	1 second
Memory limit:	256 megabytes

Sometimes Merlin might like to prove Arthur that some set S is big, but he cannot list all elements of this set because it is too big. One way to do it when membership in S can be efficiently verified is to let Arthur choose hash function h at random and some value y at random and provide him with $s \in S$ such that $h(s) = y$. With some reasonable definitions of "big" and "random hash function" this can be efficiently carried out. However, sometimes verifying that $s \in S$ can also be difficult so the following two-phase scheme is used.

Suppose, that there is a function $f(s)$. Let us say that y is k-good if the set $S_y = \{s \in S \text{ and } f(s) = y\}$ has cardinality at least k. If Merlin proves that the set G of k-good values of y is "big" — has cardinality at least d, that means that the cardinality of S is at least kd.

Consider a set S of cardinality n and a function f that has values ranging from 1 to m. We are interested in the worst case: what is the maximal z such that for any f Merlin would be able to prove that the cardinality of S is at least z using the scheme above (Merlin can choose any k and d at his discretion).

For example, let n be 5 and m be 2. Then Merlin can prove that S has at least 4 elements. Indeed: if f maps all elements of S to the same value, Merlin proves that at least 1 set is 5-good. If f maps 1 element of S to one value and 4 to another Merlin proves that at least 1 set is 4-good. If f maps 2 elements of S to one value and 3 to another, Merlin proves that at least 2 sets are 2-good. In any case he can prove that S contains at least 4 elements.

Input

Input file contains n and m ($1 \le n \le 10^{18}$, $1 \le m \le 100\,000$).

Output

Output one number z — the maximal number such that for any $f : S \to \{1, \ldots, m\}$ Merlin is able to prove that at least d values are k-good and $dk \ge z$.

Examples

`bigset.in`	`bigset.out`
`5 2`	`4`

Problem C. Coins

Input file:	`coins.in`
Output file:	`coins.out`
Time limit:	1 second
Memory limit:	256 megabytes

The group of n boys has found a treasure. The treasure consists of m golden coins. The boys decided to divide the treasure, but they couldn't decide who should take which number of coins. So they decided to ask old wise Uncle Tom to help them divide the coins.

Uncle Tom suggested the following way to divide the coins. The boys are ordered from the youngest to the oldest. First the youngest boy makes his proposal. The proposal states for each boy how many coins he should get. If all boys agree, the coins are divided according to his proposal. In the other case the second youngest boy makes his proposal, and so on.

However, such process could proceed infinitely, so Uncle Tom suggested, that if the proposal is rejected, the boys give d coins to Uncle Tom. This would stimulate the boys to make better suggestions.

The boys agreed, and ran the division. We don't ask you how many coins did Uncle Tom get, but please find out how many coins each boy would get. All the boys acted optimally in the following way: the boy agrees with the proposal that he would get x coins if and only if he would get less than x coins should this proposal be rejected.

Input

Input file contains n, m and d ($2 \le n \le 1000$, $n \le m \le 10^{18}$, $1 \le d \le 10\,000$).

Output

Output n numbers: for each boy output the number of coins he got.

Examples

`coins.in`	`coins.out`
`5 20 3`	`3 4 3 4 3`
`10 100 1`	`10 10 10 10 10 10 10 10 10 10`
`2 11 1`	`5 5`

Problem D. Duel

Input file:	`duel.in`
Output file:	`duel.out`
Time limit:	1 second
Memory limit:	256 megabytes

Alex and his rival George are preparing for the duel because of fair lady Nathalie. The duel will take place at a dark alley.

The alley has n trees and bushes growing along, the distance between adjacent plants is one meter. Alex and George decided that the duel will proceed as follows. Some tree is selected as the starting point and marked accordingly. Two trees at equal distance from the starting tree are marked as shooting points. Alex and George will start at the starting tree and move in opposite directions. When they reach shooting trees they will turn around and shoot at each other.

Given the positions of the trees, help Alex and George find the starting tree and shooting trees. First the duelists would like to know the number of ways they can choose the trees.

Input

Input file contains a non-empty string of 0-s and 1-s that describes the alley, 0 stands for the bush (that is not suitable to be neither starting nor shooting point), 1 stands for the tree. The length of the string doesn't exceed 100 000.

Output

Output the number of ways the duelists can choose starting and shooting trees.

Examples

duel.in	duel.out
`101010101`	`4`
`101001`	`0`

In the first example the following configurations of the duel are possible (starting and shooting trees are marked as bold): **1**0**1**0**1**0101, 10**1**0**1**0**1**01, 1010**1**0**1**0**1**, and **1**010**1**010**1**.

Problem E. Environment Problems

Input file:	`environment.in`
Output file:	`environment.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

A long oil pipe running along the Antarctic gulf shelf has got severe problems. Long ago it was constructed of several segments, each with length equal to one meter. Now the connections between segments are starting to wear out and oil is starting to leak to the gulf.

Antarctic Petrolium is planning to fix the problems by putting several special plastic jackets around the pipe. In process of installing the jackets the technicians would also like to check how many jackets are there around some points on the pipe.

You are the head of IT department of Antarctic Petrolium and you are asked to write a program to simulate the process. All connections of segments at the pipe are numbered from 0 to 10^9. Each jacket is described by two numbers: l_i and r_i. When it is installed, it covers all connections between l_i and r_i, inclusive.

Each request is described by one number x_i, in response to such request you must provide the number of jackets already covering the connection x_i. You must answer each request before proceeding to read the following operations.

Input

During input and output you must keep an integer value d and use it as described below. Initially d is 0.

The first line of the input file contains n — the number of operations ($1 \le n \le 200\,000$). Each of the following lines starts with a number t_i — the type of the operation, it is either 1 or 2.

If t_i is 1, the operation is installing a jacket. The description of the jacket follows. The jacket is described by two integer numbers: a_i and b_i. Use them to obtain $l_i = a_i + d$ and $r_i = b_i + d$ ($0 \le l_i, r_i \le 10^9$).

If t_i is 2, the operation is requesting the number of jackets covering some connection. The request is described by x_i. If the answer to this request is k_i, assign $d \leftarrow k_i$.

Output

For each request x_i output k_i — the number of jackets covering the connection x_i and perform assignment $d \leftarrow k_i$.

Examples

environment.in	environment.out

```
15
1 1 3
2 1
2 2
2 3
2 4
1 2 5
2 1
2 2
2 3
2 4
1 2 9
2 1
2 2
2 3
2 4
```
	```
	1
	1
	1
	0
	1
	2
	2
	1
	1
	2
	3
	2
	```

In the provided input actual jackets are $1-3$, $2-5$ and $3-10$ (when reading the last jacket description, d is 1).

Problem F. Formal Program Verification

Input file:	`formal.in`
Output file:	`formal.out`
Time limit:	1 second
Memory limit:	256 megabytes

Macrohard company is developing a new tool for formal program verification. In this tool the program is represented as a context free grammar. All procedures in the program to be verified are divided into two classes: *atomic* procedures and *complex* procedures.

Atomic procedures are procedures that do not call other procedures. Atomic procedures are denoted by lowercase letters of the English alphabet. Complex procedures can call other atomic or complex procedures. Complex procedures are denoted by uppercase letters of the English alphabet, for each complex procedure the set of *rules* is defined. Each rule has the form $A \to \alpha$ where α is a string consisting of lowercase and uppercase letters. A is called the left side of the rule, α is called the right side of the rule. The rule is interpreted in the following way: running A initiates running all procedures listed in α in order they occur in it.

The program itself is also interpreted as a complex procedure.

Running program in this model can be described in the following way. The stack of procedures is used. Initially the procedure representing the whole program is put to the stack. After that the following is repeated until the stack is empty: the procedure on the top of the stack is removed from the stack. If this procedure is atomic, it is considered to be executed and nothing else happens. If this procedure is complex, after it is executed some rule $A \to \alpha$ for this procedure is selected, and the procedures that constitute α are put to the stack. They are put to the stack in reverse order, so the first procedure in α gets to the top of the stack. The process of selecting the rule is non-deterministic, any rule for the executed complex procedure can be selected.

The process of running the program can never terminate if the stack never gets empty.

You are trying to get a job at Macrohard, and as an entrance test you are asked to create a verificator for the statement "A forces B" which is interpreted in the following way: if complex procedure A is executed, other complex procedure B is also executed when the program is run. This statement is true if whatever rules are chosen when executing complex procedures, if A is executed, B is also executed (no matter, before or after A). Note that procedure is executed when it is removed from the stack, not when it is put to it.

Input

The first line of the input file contains one uppercase letter of the English alphabet — the procedure that represents the whole program.

The second line contains n ($1 \le n \le 100$) — the number of rules. The following n lines contain rules. The right side of each rule has length at most 100. It is guaranteed that the procedure representing the whole program and each complex procedure that occurs in the right side of some rule is the left side of at least one rule.

The following line contains m ($1 \le m \le 10$) — the number of verification requests. The following m lines contain verification requests, each request is a pair of distinct uppercase letters of the English alphabet. The request to verify statement "A forces B" is specified as "A B".

Output

For each request output "`Yes`" if A forces B and "`No`" if it doesn't.

Examples

formal.in	formal.out
A	NO
5	YES
A -> a B	YES
A -> a C	NO
B -> b C	YES
C -> c	YES
C -> c B	
6	
A B	
B A	
B C	
C B	
A C	
C A	
A	NO
3	
A -> C B	
B -> b	
C -> c C	
1	
A B	

Problem G. GridBagLayout 2

Input file:	`gridbaglayout.in`
Output file:	`gridbaglayout.out`
Time limit:	1 second
Memory limit:	256 megabytes

Java user interface libraries AWT and SWING are widely used in various graphical applications. To represent the way various visual components are located in the container it uses the concept of *layout*. One of the most powerful layouts is `GridBagLayout`. In this problem we consider somewhat simplified `GridBagLayout`.

Consider a container that you consequently position components at. The container has a grid that components are arranged to row after row. Rows and columns of the grid are numbered from 0. The way that the new component is positioned on the grid is defined by the contents of special `GridBagConstraints` structure. We will consider the following fields of `GridBagConstraints`:

Field	**Definition**
`gridwidth`	Number of columns that the component occupies
`gridheight`	Number of rows that the component occupies
`gridx`	Number of leftmost column that the component occupies
`gridy`	Number of topmost row that the component occupies

The component occupies grid cells in a rectangle with leftmost column `gridx`, rightmost column `gridx + gridwidth - 1`, topmost row `gridy` and bottommost row `gridy + gridheight - 1`, with the following exceptions: `gridx` and `gridy` can be equal to a special value `GridBagConstraints.RELATIVE` and `gridwidth` and `gridheight` can be equal to a special value `GridBagConstraints.REMAINDER`.

The ways the components are placed are summarized in the following table. `REL` is used to represent value `GridBagConstraints.RELATIVE`.

`gridx`	`gridy`	**first in its row**	**Algorithm**
`REL`	`REL`	No	Find the first free cell in the topmost row of the previous component after the rightmost column of the previous component. Use this cell as the top-left cell of the component being added.
`REL`	`REL`	Yes	Find the first row after the topmost row of the previous component that has at least one free cell (if the component is the first component being added, use row 0). Find the first free cell in this row. Use this cell as the top-left cell of the component being added.
`REL`	numeric	Always	Find the first free cell in `gridy` row. Use this cell as the top-left cell of the component being added.
numeric	`REL`	No	Use the topmost row of the previous component as the topmost row of the component being added.
numeric	`REL`	Yes	Use the first row after that topmost row of the previous component that has free cell in `gridx` column as the topmost row of the component being added.

The value `gridwidth` is used to determine the number of columns, occupied by the component. If the value is equal to a special value `GridBagConstraints.REMAINDER`, the component occupies all columns to the end of each row it occupies, and the next component becomes the first component in its row.

The value `gridheight` is used to determine the number of rows, occupied by the component. If the value is equal to a special value `GridBagConstraints.REMAINDER`, the component occupies all rows to the bottom of the container in each column it occupies.

If two components intersect, or there is no required free cell in the algorithms described above, the layout is said to be invalid, and the behavior of the container is undefined.

The number of rows and columns in the grid is calculated after all components are placed, the minimal number of rows and columns required to accommodate all components is used.

Professor Dampkore doesn't like assigning absolute values to `gridx` and `gridy`, so he always uses `GridBagConstraints.RELATIVE` for their values.

The main designer of the company he works at, Dr Makbuk brings him design layout for the new program they are planning to develop. Help professor Dampkore to determine whether it is possible to create such design using `GridBagLayout` if it is not allowed to assign any values to `gridx` and `gridy` other than `GridBagConstraints.RELATIVE`.

Input

The first line of the input file must contain three integer numbers: h — the number of rows, w — the number of columns in the design, and k — the number of components in the design ($1 \le h, w \le 100$, $1 \le k \le 20$).

The following k lines contain four integer numbers each — the top row, the left column, the bottom row and the right column of each component. Components may be added in arbitrary order. Components do not overlap. All coordinates are inside the design area.

Output

If it is possible to create such design using GridBagLayout, output the program that does so, strictly adhere to the format of sample output. Use the same names for `container` and `gbc` as in sample output.

If it is impossible to do so, output "`Impossible`". Initial values of `gridwidth` and `gridheight` are 1.

Example

gridbaglayout.in

3 3 3
0 0 0 2
1 0 2 1
1 2 2 2

gridbaglayout.out

Panel container = new Panel(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();
gbc.gridwidth = GridBagConstraints.REMAINDER;
container.add(new Component(), gbc);
gbc.gridheight = 2;
gbc.gridwidth = 2;
container.add(new Component(), gbc);
gbc.gridwidth = 1;
container.add(new Component(), gbc);

gridbaglayout.in

3 3 3
0 0 0 1
1 0 2 1
1 2 2 2

gridbaglayout.out
Impossible

Problem H. High School Duels

Input file:	`high.in`
Output file:	`high.out`
Time limit:	1 second
Memory limit:	256 megabytes

Retrozavodsk State University has decided to run a new type of High School programming contests they call duel series. Each contest consists of a series of matches with two teams participating in each match. One problem is proposed at each match and the team that solves the problem first is considered the winner of the match. The time used to solve the problem is measured in nanoseconds, so there are no ties. During the contest each team plays several matches and the team that wins strictly more matches than any other team is considered the winner of the contest. If there is no such team, several teams that solve most problems tie for the first place and are all considered winners.

After the part of the contest has passed, professor Smithoff, the organizer of the contest, started to wonder which teams could still win the contest, or at least tie for the first place. For each team he knows that the number of its wins, and for each pair of teams he knows the number of duels this pair of teams still has to play.

Ogel Rekhristo, the statistics manager of the contest has presented the list of teams that can win to professor Smithoff. However, the professor is not satisfied — he would like to know the reason the teams cannot win.

Consider team x and a set Y of teams ($x \notin Y$) such that the total number of wins of teams from Y plus the total number of duels they should play among each other is s. If $s/|Y|$ is greater than the number of wins of team x plus the number of duels x still has to play, x clearly cannot win. Such set is called the blocking set for x. It turned out that if the team x cannot win the contest it is always because of some such blocking set Y.

Help Ogel to find out which teams can win and find blocking set for each team that cannot win.

Input

The first line of the input file contains n — the number of teams ($2 \le n \le 30$). The following n lines contain team descriptions, each team is described by its name (a word containing letters of the English alphabet, numbers and underscores) followed by the number of its wins w ($0 \le w \le 1000$).

The following n lines contain n integer numbers each, the j-th number in the i-th line is $d_{i,j}$ — the number of duels that the i-th team has to play versus the j-th team ($0 \le d_{i,j} \le 1000$, $d_{i,i} = 0$, $d_{i,j} = d_{j,i}$).

Output

For each team output either that it can win the contest, or that it cannot win, in the latter case output the blocking set for the team. If there are several blocking sets, output any one. Strictly adhere to the format of sample output.

Examples

high.in
```
4
PetrozavodskSU 13
SPbIFMO 11
MSU 10
SaratovSU 7
0 1 6 1
1 0 0 2
6 0 0 0
1 2 0 0
```

high.out
```
PetrozavodskSU can win
SPbIFMO cannot win because of the following teams:
  PetrozavodskSU, MSU
MSU can win
SaratovSU cannot win because of the following teams:
  PetrozavodskSU
```

Consider, for example, SPbIFMO team. It has 11 wins and 3 matches left. So it can have at most 14 wins. But PetrozavodskSU and MSU have a total of 23 wins and 6 matches left. So their total number of wins will be 29, 14.5 on average per team. So at least one team from this set will have more than 14 wins in the end, and SPbIFMO team cannot win.

Problem I. Impossible to Solve

Input file:	`impossible.in`
Output file:	`impossible.out`
Time limit:	3 seconds
Memory limit:	256 megabytes

Programmers say that this problem is impossible to solve. However, linguists and archeologists solve even more difficult cryptoanalytic problems every day, so they would probably consider this problem easy.

Consider some book, written in English. Denote its text as S. Process S in the following way: replace all characters with ASCII code outside of 32..127 range with space character (ASCII code 32). Call the resulting text X.

Choose n from 2 to 20, choose random permutation π of numbers from 1 to n. Split X to blocks $X_1, X_2, \ldots, X_t$ of length n (the last block X_t can have length less than n). After that reorder characters in each X_i using permutation π (the last block X_t is reordered only if its length is n). For example, if X_i is "`Contest`" and $\pi = \langle 2, 3, 1, 7, 4, 5, 6 \rangle$ the result of reordering is "`onCttes`". Concatenate the resulting blocks to get the new text Y.

Choose m from 2 to 20, choose random sequence K of length m, let K contain numbers from 0 to 127. Split Y to blocks $Y_1, Y_2, \ldots, Y_r$ of length m (the last block Y_r can have length less than m). After that apply bitwise `xor` to each block Y_i and K (the ASCII code of j-th character of Y_i is xor-ed with j-th number of K and the character with the corresponding ASCII code is used). For the last block Y_r the first $|Y_r|$ elements of K are used. For example, if Y_i is "`Contest`" and $K = \langle 1, 2, 1, 2, 1, 3, 0 \rangle$, the resulting of xor-ing is "`Bmovdpt`". Concatenate the resulting blocks to get the new text Z.

You are given Z, you have to restore X.

Input

Input file is binary in this problem. It has size from 140 to 650 kilobytes. It was created by applying the described procedure to some classic English literature text downloaded from Project Gutenberg web site `http://www.gutenberg.org/`.

Output

Output text file containing X.

Examples

You can find sample input and sample output in the zip archive with the statement.

Problem J. Jackpot

Input file:	`jackpot.in`
Output file:	`jackpot.out`
Time limit:	1 second
Memory limit:	256 megabytes

Alice and Bob are finalists of the TV show "Lucky Loser". In order to find out who of them would win the main jackpot they are suggested to play the following game.

They are given an undirected graph G with n vertices numbered from 1 to n. Initially all vertices of the graph are uncolored.

Players make moves in turn, Alice moves first. On the i-th move (its Alice's move if i is odd, and Bob's move if i is even) the player must color the i-th vertex of the graph either black or white. If after the player's move there is an edge in the graph that connects vertices of the same color, the player whose move it was loses the game and the other player wins the jackpot. If no player loses the game and the whole graph is colored, the game is declared to end in a tie and the players divide jackpot between them.

Help the judges of the show to understand who would win the game. If the player X wins the game, also find the minimal number of the move i such that regardless of the moves of the other player, the player X can always win after the move i.

Input

The first line of the input file contains n and m — the number of vertices and edges of the graph, respectively ($1 \le n \le 100\,000$, $0 \le m \le 100\,000$).

The following m lines describe edges of the graph, the edge is described with its endpoints. There are no loops and no parallel edges.

Output

Output the name of the player who wins the game followed by the number of move after which it certainly happens. If there is a tie, output "`Tie`".

Examples

jackpot.in	jackpot.out

3 2
1 3
2 3
	Bob 3

4 2
1 4
3 4
	Alice 4

3 2
1 2
2 3
	Tie