Problem A. Graph Game

Input file:	`game.in`
Output file:	`game.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Nick and Peter like to play the following game when attending their complexity theory lectures. They draw an undirected bipartite graph G on a sheet of paper, and put a token to one of its vertices. After that they make moves in turn. Nick moves first.

A move consists of moving the token along the graph edge. After it the vertex where the token was before the move, together with all edges incident to it, are removed from the graph. The player who has no valid moves loses the game.

You are given the graph that Nick and Peter have drawn. For each vertex of the graph find out who wins if the token is initially placed in that vertex. Assume that both Nick and Peter play optimally.

Input

The first line of the input file contains three integer numbers n_1, n_2, and m — the number of vertices in each part, and the number of edges, respectively ($1 \le n_1, n_2 \le 500$, $0 \le m \le 50\,000$). The following m lines describe edges — each line contains the numbers of vertices connected by the corresponding edge. Vertices in each part are numbered independently, starting from 1.

Output

Output two lines. The first line must contain n_1 characters, the i-th character must be '`N`' in case Nick wins if the token is initially placed in the i-th vertex of the first part, and '`P`' if Peter does. The second line must contain n_2 characters and describe the second part in the same way.

Example

game.in	game.out
3 3 5 1 1 1 2 1 3 2 1 3 1	NPP NPP

Problem B. Lempel-Ziv Compression

Input file:	`lz.in`
Output file:	`lz.out`
Time limit:	1 second
Memory limit:	64 megabytes

Most modern archivers, such as WinRAR or WinZIP, use modifications of Lempel-Ziv method as their primary compression algorithm. Although decompression of LZ-compressed archives is usually easy and fast, the compression process itself is often rather complicated and slow. Therefore professional archivers use approximation methods that sometimes do not allow to achieve the best possible compression.

This situation doesn't satisfy your chief George. He would like to create the best archiver WinGOR. The archiver will use the following modification of LZ77 algorithm.

The text is partitioned to chunks of length not exceeding 4096. Each chunk is compressed independently. We will describe the decompression of one chunk t. Based on this description, you will have to create a compression algorithm that will create the shortest possible compressed chunk x from the given chunk t.

The compressed chunk is written down as the sequence of *plain characters* and *repetition blocks*. Plain character is 8 bits long. When decompressing, plain character c is simply copied to output. Repetition block (r, l) consists of two parts: *reference* r and *length* l, each 12 bits long. Reference r is an integer number between 1 and 4095. When repetition block (r, l) is obtained after decompressing $i-1$ characters of text, characters $t[i-r \ldots i-r+l-1]$ are copied to output. Note, that r can be less than l, in this case recently copied characters are copied to output as well.

To help decompressor distinguish between plain characters and repetition blocks a leading bit is prepended to each element of the compressed text: 0 means plain character follows, 1 — repetition block follows.

For example, "`aaabbaaababababababab`" can be compressed as "`aaabb(5,4)(2,10)`". The compressed variant has $8+8+8+8+8+24+24+7=95$ bits instead of 152 in the original text (additional 7 bits are used to distinguish between plain characters and repetition blocks).

Given a text chunk, find its compressed representation which needs fewest number of bits to encode.

Input

Input file contains a text chunk t. Its length doesn't exceed 4096. A text chunk contains only small letters of the English alphabet.

Output

Print the length of the compressed text in bits at the first line of the output file. Print the compressed chunk itself at the second line of the output file. Use characters themselves to denote plain characters and "`(r,l)`" notation (without spaces) to denote repetition blocks.

Example

lz.in	lz.out
aaabbaaababababababab	95 aaabb(5,4)(2,10)

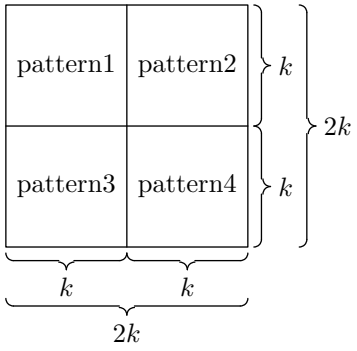
Problem C. Map Generator

Input file:	`map.in`
Output file:	`map.out`
Time limit:	1 second
Memory limit:	64 megabytes

Peter is writing the random map generator for the new game *Heroes of Mouse and Keyboard*. The map for the game has the structure of square grid of size $2^m \times 2^m$, each unit square of which may be filled with either land, or water.

The Peter's generator creates the map description. The map description has the following structure. Initially there are two patterns: '`1`' and '`0`', standing for a unit square of land and water, respectively. These patterns have size 1. The description of the map is the sequence of rules of the following form: "`<pattern>=<pattern1>,<pattern2>,<pattern3>,<pattern4>`"

Here `<pattern`i`>` are names of already declared patterns of some size k. After this rule is processed, the new pattern `<pattern>` becomes known, which has the size $2k$. This pattern is constructed by putting the patterns from the right side of the rule together to form a square of size $2k \times 2k$, as shown on the picture below.



Now Peter has generated the map and wonders how many connected components of land are there. Two land squares belong to the same connected component if they have a common side. Help him!

Input

The input file contains the description of the map. The names of the patterns consist of English letters and digits only. All names are case-sensitive, their lengths doen't exceed 20 characters. No two names coincide. The maximal size of the pattern is $2^{16} \times 2^{16}$. The number of patterns doesn't exceed 200.

The map itself is specified by the pattern with the special name "`Map`". This pattern is described last. There are no unneeded patterns, i.e. each pattern is used to create map (except, possibly, "`0`" or "`1`").

Output

Output one number — the number of connected components in the map.

Example

map.in	map.out
A=0,1,1,0 X=1,1,1,1 B=A,A,A,X C=B,B,B,B Map=C,C,C,C	56

Problem D. Mean Payoff Game

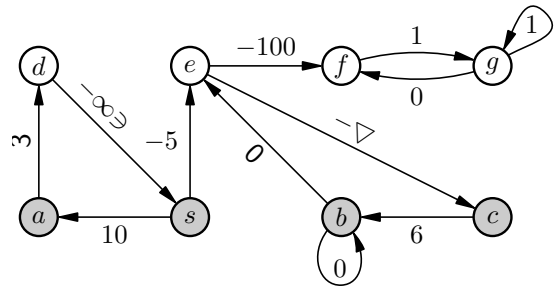
Input file:	`mean.in`
Output file:	`mean.out`
Time limit:	3 seconds
Memory limit:	64 megabytes

Mean payoff game is a two-player antagonistic game which is played on a directed graph. The vertices of the graph are divided into two sets A and B. The first player owns vertices from the set A, and the second player owns the vertices from the set B. Each arc uv of the graph is marked with some integer number w_{uv}.

The game proceeds as follows. Initially the token is placed in some vertex s of the graph. After that players move the token along arcs of the graph. If the token is placed in a vertex from the set A, the first player moves the token, if it is placed in a vertex from the set B, the second player does. Players maintain a common counter z which accumulates the sum of numbers written on the arcs passed by the token. Initially $z = 0$, after the token is moved along the arc uv, the assignment $z \leftarrow z + w_{uv}$ is performed.

The purpose of the first player is to make $z \to +\infty$. The purpose of the second player is to make $z \to -\infty$. If none of the players can fulfil his goal, the game is draw.

For example, let us consider the game shown on the picture below. Shaded vertices belong to the first player, white ones belong to the second player. Initially the token is in the vertex s. This game is a victory for the first player. She must move a token to vertex e, and whatever move makes the second player, he either enters a positive cycle $f \to g \to f$ with no exit, or moves the token to vertex c, from which the first player moves it to b, and then back to e, adding a total of $+1$ to z.



If can be proved that there exists a *deterministic strategy* for a winning player. This strategy specifies for each vertex in a player's set, what arc should she move a token along to win. Such strategy depends on neither current value of z, nor on any other information on the previous play.

Alice is going to play the game with Bob. Alice is the first player. She thinks that she wins the game, and has developed a winning strategy. But she is not absolutely sure. She asks you to tell whether her strategy is actually a winning strategy for the first player — that is, whether regardless of Bob's moves Alice can make $z \to +\infty$ following her strategy.

Input

The first line of the input file contains two integer numbers n and m — the number of vertices and edges in the graph, respectively ($1 \le n \le 2\,000$, $1 \le m \le 10\,000$). The second line contains n characters that describe the ownership of the vertices — the vertices that belong to Alice are marked with '`A`', the vertices that belong to Bob are marked with '`B`'. The third line contains s — the number of the starting vertex.

The following m lines describe the arcs of the graph. Each arc is specified by the vertex it starts at, the

vertex it leads to, and the number written on it. Numbers on arcs do not exceed 10^5 by their absolute values. Each vertex has at least one arc leaving it.

The last line describes Alice's strategy. It contains n integer numbers. For each vertex the corresponding number is either 0, if the vertex belongs to Bob, or is a 1-based number of the arc that Alice must move the token along from this vertex due to her strategy.

Output

If Alice's strategy is actually a winning strategy, print "`Win`" at the first line of the output file. In the other case print either "`Lose`" if Bob can force Alice to lose if she follows this strategy, or "`Draw`" if Bob cannot force her lose, but can force a draw. In these cases the second line must describe Bob's strategy. It must contain n integer number. For each vertex the corresponding number must be either 0, if the vertex belongs to Alice, or a 1-based number of the arc that Bob must move the token along from this vertex due to his strategy.

Example

mean.in	mean.out
8 12 AAAABBBB 2 2 1 10 3 3 0 4 3 6 1 5 3 5 2 -13 2 6 -5 3 6 0 6 4 -5 6 7 -100 7 8 1 8 7 0 8 8 1 4 6 7 3 0 0 0 0	Win
8 12 AAAABBBB 2 2 1 10 3 3 0 4 3 6 1 5 3 5 2 -13 2 6 -5 3 6 0 6 4 -5 6 7 -100 7 8 1 8 7 0 8 8 1 4 1 7 3 0 0 0 0	Draw 0 0 0 0 5 8 10 11

Problem E. Median

Input file:	`median.in`
Output file:	`median.out`
Time limit:	1 second
Memory limit:	64 megabytes

Boolean functions form one of the central topics of discrete mathematics. A set S of boolean functions is said to *be complete* or form a *basis* if any boolean function can be represented as a composition of functions from a set S.

There are classical bases, such as $\{\vee, \wedge, \neg\}$ for disjunctive or conjuctive normal forms, $\{\oplus, \wedge, 1\}$ for Zhegalkin polynomials, as well as nice one-function bases, such as Peirce arrow $\downarrow$ (not or), or Sheffer stroke $'$ (not and).

Most bases that are usually considered involve binary or unary operations. However, there are several ternary operations that are also very important. One of these operations is *median*: $\langle xyz \rangle$ is equal to 0 if at least two arguments are 0, or 1 if at least two arguments are 1. It is easy to represent it using conjunction and disjunction: $\langle xyz \rangle = (x \wedge y) \vee (y \wedge z) \vee (z \wedge x) = (x \vee y) \wedge (y \vee z) \wedge (z \vee x)$.

A function $f(x_1, x_2, \ldots, x_n)$ is said to be *self-dual* if for any set of variables the following condition is satisfied: $f(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}) = \overline{f(x_1, x_2, \ldots, x_n)}$ ($\overline{x}$ means "not x"). An example of a self-dual function is $f(x, y, z) = x \oplus y \oplus z$.

A function $f(x_1, x_2, \ldots, x_n)$ is said to be monotone if for any two sets of variables $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ such that $x_i \leq y_i$ for all i, the following condition is satisfied: $f(x_1, x_2, \ldots, x_n) \leq f(y_1, y_2, \ldots, y_n)$. An example of a monotone function is $f(x, y, z) = x \wedge y \wedge z$.

It is easy to check that median is both self-dual and monotone. More of that, median is complete for the class of self-dual monotone functions. It means that every self-dual monotone function can be represented as a composition of medians. For example, the function $f(u, v, x, y, z) = ((u \vee v \vee x \vee y) \wedge z) \vee (u \wedge v \wedge x \wedge y)$ can be represented as $f(u, v, x, y, z) = \langle uz \langle vz \langle xzy \rangle \rangle \rangle$.

Given a self-dual monotone boolean function, find its representation with median operations.

Input

The input file contains a formula of a self-dual monotone boolean function. The formula uses small letters of the English alphabet for variables, and the following operations (in order of precedence): '`!`' for `not`, '`&`' for `and`, '`|`' for `or`. Parenthesis change precedence as usually. The formula contains at most 5 variables. The length of the formula doesn't exceed 250 characters.

Output

Output the representation of the formula from the input file with median operations. Do not output spaces. Your formula must not contain more than 50 000 characters.

Example

median.in	median.out
`x&y|y&z|x&z`	`<xyz>`
`x`	`x`
`((u|v|x|y)&z)|(u&v&x&y)`	`<uz<vz<xzy>>>`

Problem F. Money, Money, Money

Input file:	`money.in`
Output file:	`money.out`
Time limit:	1 second
Memory limit:	64 megabytes

The government of Flatland has decided to carry out the money system reform. The purpose of the reform is to reduce the number of different banknotes denominations down to two. After the reform there will be two types of banknotes — a tupiks and b tupiks.

The problem is that the president of Flatland doesn't like the number x. Therefore the minister of finances was instructed to choose such a and b that it is impossible to pay exactly x tupiks without change. On the other hand it must be possible to pay all amounts larger than x.

Now you are asked to help him — choose such a and b, or recommend the minister to retire, if it is impossible.

Input

Input file contains one number x ($1 \le x \le 10^{12}$).

Output

Output two integer numbers a and b, such that it is impossible to pay x tupiks using banknotes of a and b tupiks without change, but it is possible to pay any larger sum.

If it is impossible, output two zeroes to the output file.

Example

money.in	money.out
3	2 5
4	0 0
5	3 4

Problem G. Musical

Input file:	`musical.in`
Output file:	`musical.out`
Time limit:	1 second
Memory limit:	64 megabytes

Witty likes musicals most of all. He have seen all musicals in the world, and have seen some of them for ten or more times. But now the great time has come for Witty. He has put his own musical on a stage!

Now he plans a tour around the country with his musical. The tour will visit n cities. Each week the musical will be staged in some new city. There will be a performance every day.

Witty must choose the order in which the cities should be visited, to maximize the profit. Some cities are connected by bidirectional roads. Although musical equipment and performers move from city to city by air, roads play important role in a way musical fans visit musicals and exchange information about musicals.

To estimate the profit Witty has gathered information about musical fans in all cities. He divided them to three categories.

- Ordinary musical fans go to one musical per week. They only attend musicals that are staged in their city. When there are several musicals possible, they choose the one which they have most information about. The amount of information depends on the number of neighboring cities that were already visited by the musical. Two cities are neighboring if they are connected by a road. If there are several musicals they have equal information about they go to a random one.
- Crazy musical fans go to one musical every day. When there are several musicals in the city, they go to a random one.
- Finally, musical maniacs go to one musical every day, choosing one from the city they live in and all neighboring cities. When there are several possible musicals to choose from, they choose a random one.

Having all this information, as well as the information about all other musicals that will be staged during next weeks, help Witty to choose the order in which the cities should be visited by the tour to maximize the expected number of fans that would come to see his musical.

Input

The first line of the input file contains three integer numbers: n, m and k — the number of cities, roads, and other musicals ($1 \le n \le 10$, $0 \le m$, $0 \le k \le 10$). The following n lines describe cities. Each city is described with three integer numbers not exceeding 10^6 — the number of ordinary musical fans, the number of crazy musical fans, and the number of musical maniacs.

The following m lines describe roads, each road is described by two integer numbers — the numbers of cities it connects. No two cities are connected by more than one road, no road connects a city to itself.

The following k lines give the time table of other musicals. Each musical is described with n numbers, the i-th of these numbers is the number of the city where the musical will be staged on the i-th week, or 0 if the musical will be on vacation that week. Note that unlike Witty's musical, other musicals may perform in the same city for more than once.

Output

Print the maximal possible expected amount of attendants of Witty's musical at the first line of the output file. Your answer must be accurate up to 10^{-6}.

The second line must contain the permutation of n numbers — the order in which the cities must be visited to maximize the expected attendance.

Example

musical.in	musical.out
3 2 1 100 10 20 20 50 30 10 40 30 1 2 1 3 2 3 1	1670.00000000 1 2 3

In this example, there are 555 people expected to come see musical on the first week — 100 ordinary fans, 10 crazy fans every day for a total of 70, $20/2 = 10$ maniacs from the first city, $30/2 = 15$ maniacs from the second city, and 30 maniacs from the third city every day for a total of $(10 + 15 + 30) \cdot 7 = 385$ maniacs, and therefore $100 + 70 + 385 = 555$ people on the first week.

On the second week the musical is expected to be visited by $20 + 50 \cdot 7 + (20/2 + 30) \cdot 7 = 650$ people, and on the third week by $10 + 40 \cdot 7 + (20/2 + 30/2) \cdot 7 = 465$ people, for a total of 1670 expected people on all three weeks.

Problem H. Polygon

Input file:	`polygon.in`
Output file:	`polygon.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The Ministry of Defense of Flatland is planning to build a new polygon. The polygon must have a form of a perfect circle.

However, generals in the Ministry are concerned about the security of the polygon. There are several special power shields above Flatland, each of them has a form of a rectangle with edges parallel to coordinate axes. The generals want to find such a place for the polygon, that there are two different power shields that completely cover the polygon.

Of course, the generals want to build as large polygon as possible. Help them to find such position for the polygon that it has the greatest possible area.

Input

The first line of the input file contains n — the number of the power shields ($1 \le n \le 60\,000$). The following n lines describe shields — each shield is described with four integer numbers: x_{min}, y_{min}, x_{max}, y_{max}. All coordinates do not exceed 100 000 by their absolute values.

Output

Output three real numbers — the coordinates of the center of the polygon to build and its radius. All numbers must be printed with exactly one digit after the decimal point.

If it is impossible to build the polygon, print "`Impossible`" at the first line of the output file.

Example

polygon.in	polygon.out
4 0 0 2 3 1 -1 4 1 1 1 4 4 2 0 5 5	3.0 2.0 1.0
1 0 0 1 1	Impossible
2 0 0 3 3 0 0 3 3	1.5 1.5 1.5

Problem I. Protect the Trees

Input file:	`protect.in`
Output file:	`protect.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Johnny has got a birthday present — a little puppy. He loves his puppy very much. He called him Merry. Merry is very little, but he's already got his teeth. They are very sharp, and must be growing because he likes to gnaw everything that comes to his sight.

Merry lives in Johnny's backyard. But there is one problem. Johnny's father — a botanist — has grown a couple of rare trees there. Now he is afraid that Merry can hurt their delicate bark. So Johnny is planning to protect the trees.

He has found three long planks at his old workshop, so now is going to surround his trees by a triangular fence. No part of the plank must be outside the fence, because somebody could step upon it and fall down.

It may turn out that it is impossible to surround all the trees. In this case Johnny wants to surround as many trees as possible. His father is still planning to use planks in some other way afterwards, so Johnny mustn't cut them to pieces.

Help Johnny to find out how many trees he can save.

Input

The first line of the input file contains n — the number of father's trees in Johnny's backyard ($1 \le n \le 70$). The second line contains three integer numbers a, b and c — the lengths of planks Johnny's father has got ($1 \le a, b, c \le 10^4$, a, b and c can be sides of a non-degenerate triangle). The following n lines contain two integer numbers each and describe trees. Each tree is described by its coordinates, they do not exceed 10^4 by their absolute values. You can neglect the trees thickness. No two trees coincide.

Output

Output one integer number — the number of the trees that can be saved.

Example

protect.in	protect.out
5 3 4 6 6 0 0 0 1 2 2 3 6 1	3

Problem J. 2-3 Trees

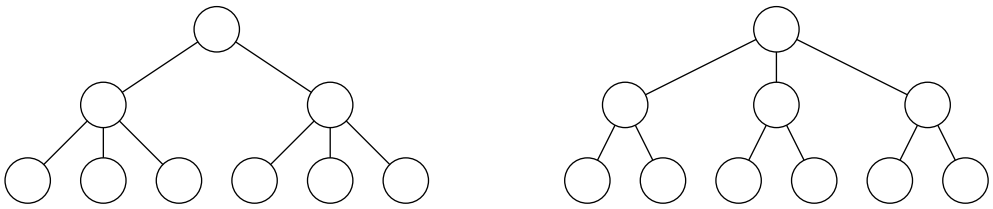
Input file:	`twothree.in`
Output file:	`twothree.out`
Time limit:	1 second
Memory limit:	64 megabytes

2-3 tree is an elegant data structure invented by John Hopcroft. It is designed to implement the same functionality as the binary search tree. 2-3 tree is an ordered rooted tree with the following properties:

- the root and each internal vertex have either 2 or 3 children;
- the distance from the root to any leaf of the tree is the same.

The only exception is the tree that contains exactly one vertex — in this case the root of the tree is the only vertex, and it is simultaneously a leaf, i.e. has no children. The main idea of the described properties is that the tree with l leaves has the height $O(\log l)$.

Given the number of leaves l there can be several valid 2-3 trees that have l leaves. For example, the picture below shows the two possible 2-3 trees with exactly 6 leaves.



Given l find the number of different 2-3 trees that have l leaves. Since this number can be quite large, output it modulo r.

Input

Input file contains two integer numbers: l and r ($1 \le l \le 5\,000$, $1 \le r \le 10^9$).

Output

Output one number — the number of different 2-3 trees with exactly l leaves modulo r.

Example

twothree.in	twothree.out
6 1000000000	2
7 1000000000	3