Problem A. Easy-to-Solve Expressions

Source file name:	Expressions.c, Expressions.cpp, Expressions.java, Expressions.py
Input:	`Standard`
Output:	`Standard`

When one looks at a set of numbers, one usually wonders if there is a relationship among them? This task is more manageable if there are only three numbers.

Given three distinct positive integers, you are to determine how one can be computed using the other two. Print 1 if any of the three numbers is the *sum* of the other two numbers, print 2 if any of the three numbers is the *product* of the other two numbers, print 3 otherwise. Assume that exactly one of these three messages will apply.

Input

There is only one input line; it contains three distinct positive integers, each between 2 and 1000 (inclusive).

Output

Print the appropriate message as described above.

Example

Input	Output
10 30 20	1
10 20 200	2
100 5 700	3

Problem B. Easy-to-Pronounce Words

Source file name:	Words.c, Words.cpp, Words.java, Words.py
Input:	`Standard`
Output:	`Standard`

We define a word as *easy-to-pronounce* if every vowel in the word is immediately followed by a consonant and every consonant in the word is immediately followed by a vowel. The first letter of the word can be a vowel or consonant. Assume that the vowels are: `a`, `e`, `i`, `o`, `u` (note that the letter `y` is not a vowel, i.e., it is a consonant).

Given a word, print 1 (one) if it is easy-to-pronounce, 0 (zero) otherwise.

Input

There is only one input line; it contains a word consisting of 1-30 lowercase letters (starting in column 1). Assume that there will not be any characters other than the lowercase letters in the input.

Output

Print the appropriate message as described above.

Example

Input	Output
contest	0
coaches	0
cocahes	1
ali	1

Problem C. Decimal XOR

Source file name:	Decimalxor.c, Decimalxor.cpp, Decimalxor.java, Decimalxor.py
Input:	`Standard`
Output:	`Standard`

The binary operation **XOR** accepts two binary digits as input and outputs a binary digit: if both input digits are 0 (or both are 1), the output is 0; otherwise the output is 1. We can look at this as: if both input values are `low` (or both are `high`), the output is 0; otherwise the output is 1.

Decimal numbers have several digits and each digit can be one of 10 values (0-9). We define the operation **DEXOR** (**XOR** of two decimal numbers) as follows: we **DEXOR** two decimal digits at a time; the two decimal digits at 1^{st} position are **DEXOR**'ed, the two decimal digits at 10^{th} position are **DEXOR**'ed, the two digits at 100^{th} position are **DEXOR**'ed, etc. When **DEXOR**'ing two decimal digits, the result digit is 0 if both digits are too small (≤ 2) or both digits are too large (≥ 7); the result digit is 9 otherwise.

Given two decimal numbers, compute their **DEXOR**.

Input

There are two input lines, each line providing a decimal number between 0 and $999,999$ (inclusive). Assume that there will not be extra leading zeroes in an input number, i.e., there will not be extra zeroes at the beginning of a number in the input.

Output

Print the **DEXOR** of the two decimal numbers. When **DEXOR**'ing two decimal numbers, if one has fewer digits, it should be considered as having zeros on the left to make both numbers having the same number of digits. The result should have as many digits as the larger number.

Example

Input	Output
22776 15954	09099
29 18908	09900

Explanation

Note that, in the second Example, 29 should be treated as 00029 so that it will have the same number of digits as the second number (so that they can be **DEXOR**'ed digit-by-digit).

Problem D. Square Fishing Net

Source file name:	Fishing.c, Fishing.cpp, Fishing.java, Fishing.py
Input:	`Standard`
Output:	`Standard`

With so many activities/events being virtual these days, we are going on a virtual fishing trip!

Given the $(x,\ y)$ coordinates of n points (each point represents a fish) and a square (representing a fishing net), what is the maximum fish you can catch with one try? You can place the square net anywhere but its sides must be parallel to X-axis and Y-axis. A fish is caught if it is inside or on the boundary of the net.

Input

The first input line contains two integers: s ($1 \leq s \leq 100$), indicating the length of one side of the fishing net and n ($1 \leq n \leq 100$), indicating the number of fish. Each of the next n input lines contains two integers (each between 1 and 100, inclusive) indicating the $(x,\ y)$ coordinates of one fish. Assume that no two fish are at the same location.

Output

Print the maximum number of fish you can catch.

Example

Input	Output
3 8 2 1 2 3 5 1 5 2 3 2 4 2 10 5 11 5	6
50 2 10 5 11 5	2

Problem E. Aqualin

Source file name:	Aqualin.c, Aqualin.cpp, Aqualin.java, Aqualin.py
Input:	`Standard`
Output:	`Standard`

The board game Aqualin is played on an $n \times n$ grid, where each grid cell is filled with a piece. Each piece is an animal of a particular color, i.e., each piece represents two properties: animal type and animal color. For simplicity, we assign letters '`A`' through '`Z`' for animal types and number the different colors 1 to n.

After each of the n^2 cells of the grid are filled, the game is scored. There are two teams.

The first team gets points for the largest connected component of animals of the same type that are size 2 or greater. Specifically, for each of the largest connected components of the same type of animals, the first player gets $1 + 2 + \ldots + (c - 1)$ points, where c is the number of animals in the component. For example, if there are connected components of 3 starfish ('`A`'), 4 octopi ('`B`'), 1 whale ('`C`'), and 2 starfish ('`A`'), then points would be awarded for only the first two groups of animals. **Note that no score is awarded to a component of size 1 and, when there are several connected components of the same animal type, only the largest connected component is awarded points.** Thus, in the case discussed above, the first team would get $1 + 2 = 3$ points for the starfish, and $1 + 2 + 3 = 6$ points for the octopi, for a total of 9 points. A connected component of animals is the set of animals you can reach directly or indirectly by traveling up, down, left or right in the grid by only going through animals of the same type.

The second team gets points for the largest connected component of animals of the same color. The scoring is the same as previously described, based on component size.

Here is an example 5×5 grid, filled out. In each cell, an ordered pair $(x,\ y)$ indicates that the animal in that cell is of type x, color y.

(B,3)	(A,1)	(C,1)	(A,2)	(A,5)
(B,4)	(B,1)	(B,5)	(E,4)	(E,3)
(C,3)	(C,2)	(B,2)	(D,2)	(E,2)
(A,3)	(C,4)	(A,4)	(E,5)	(D,1)
(D,3)	(C,5)	(D,4)	(D,5)	(E,1)

First, let's consider the score by animal type. There are two animals of type '`A`' in the top right corner for a score of 1. All five animals of type '`B`' are connected (look at top left) for a score of 10. Four animals of type '`C`' are connected, starting from the animal (`C`, 3) for a score of 6. Two animals of type '`D`' are connected, (`D`, 4) and (`D`, 5), for a score of 1. Three animals of type '`E`' are connected, (`E`, 4), (`E`, 3) and (`E`, 2), for a score of 3. The total score for the first team is $1 + 10 + 6 + 1 + 3 = 21$.

There are three animals of color 1 connected at the top for a score of 3. Note that there are 2 other animals of color 1 connected in the bottom right corner, but this score doesn't count because we only count the largest connected component of a single type (or color) of an animal. There are 4 animals of color 2 connected (all on row 3) for a score of 6. There are 3 animals of color 3 connected (all on column 1) for a score of 3. There are 3 animals of color 4 connected, (`C`, 4), (`A`, 4) and (`D`, 4), for a score of 3. There are 2 animals of color 5 connected, (`E`, 5) and (`D`, 5), for a score of 1. The total score for the second team is $3 + 6 + 3 + 3 + 1 = 16$.

Note: In the given example, there are five animal types and five animal colors. Although each of the 25

possible animal type/color combinations appeared exactly once in this example, this is not guaranteed to be true for all input grids. That is, some animal type/color combinations may appear more than once and some animal type/color combinations may not appear at all.

Given the contents of the grid at the end of a game, determine the score for both teams.

Input

The first input line contains a single integer: n ($2 \leq n \leq 26$), indicating the number of rows and columns in the game grid. Each of the following n input lines provides the contents of one row in the grid. Each row is represented by n terms, each term providing the type x ('A' $\leq x \leq$ 'Z') and color y ($1 \leq y \leq n$) of the corresponding animal on the grid. Assume that the input rows start in column one and there is exactly one space separating different values on these input lines.

Output

Print, on a line by itself, the score for the first team, followed by a space, followed by the score for the second team.

Example

Input	Output
5 B 3 A 1 C 1 A 2 A 5 B 4 B 1 B 5 E 4 E 3 C 3 C 2 B 2 D 2 E 2 A 3 C 4 A 4 E 5 D 1 D 3 C 5 D 4 D 5 E 1	21 16
2 A 1 B 1 A 1 B 1	2 6

Problem F. RCV Simplification

Source file name:	RCV.c, RCV.cpp, RCV.java, RCV.py
Input:	`Standard`
Output:	`Standard`

The following is from Ballotpedia [https://ballotpedia.org/Ranked-choice_voting_(RCV)]:

Broadly speaking, the *ranked-choice voting process* unfolds as follows for single-winner elections:

1. Voters rank the candidates for a given office by preference on their ballots.
2. If a candidate wins an outright majority of first-preference votes (i.e., 50 percent plus one), he or she will be declared the winner.
3. If, on the other hand, no candidates win an outright majority of first-preference votes, the candidate with the fewest first-preference votes is eliminated.
4. All first-preference votes for the failed candidate are eliminated, lifting the second-preference choices indicated on those ballots.
5. A new tally is conducted to determine whether any candidate has won an outright majority of the adjusted voters.
6. The process is repeated until a candidate wins a majority of votes cast.

Example: Assume that there are four candidates in an election. The table below presents the raw first-preference vote totals for each candidate:

Raw first-preference vote tallies		
Candidate	**First-Preference Votes**	**Percentage**
Candidate A	475	46.34%
Candidate B	300	29.27%
Candidate C	175	17.07%
Candidate D	75	7.32%

In the above scenario, no candidate won an outright majority of first-preference votes. As a result, the candidate (Candidate D) with the smallest number of first-preference votes is eliminated. The ballots that listed candidate D as the first preference are adjusted, raising their second-preference candidates. Assume that, of the 75 first-preference votes for Candidate D, 50 listed Candidate A as their second preference and 25 listed Candidate B. The adjusted vote totals would be as follows:

Adjusted vote tallies		
Candidate	**Adjusted First-Preference Votes**	**Percentage**
Candidate A	525	51.22%
Candidate B	325	31.71%
Candidate C	175	17.07%

On the second tally, Candidate A secured 51.22 percent of the vote, thereby winning the election.

Note: If several candidates are tied for the fewest first-preference votes, all such candidates are eliminated. So, candidates not eliminated must have at least one more first-preference vote than those eliminated.

We have received information on the percentage for the first-preference for each candidate, but we don't know how the candidates are listed as the second preference, third preference, etc. Help write a program to remove candidates that cannot possibly win. More specifically, given the current votes for a set of candidates, find the set of candidates that cannot possibly win.

Input

The first input line contains a single integer, N ($1 \leq N \leq 10^5$), representing the number of votes. Each of the following N input lines contains a candidate name receiving the first-place vote from that voter. Each candidate name is 1-25 letters (lowercase and uppercase), starting in column one.

Output

On the first output line, print a single positive integer, C, the number of candidates that cannot win. Each of the remaining C output lines should contain a candidate name that cannot win. The candidate names should be printed in lexicographical order (increasing order).

Example

Input	Output
5 Alice Alice Bob Bob Carol	1 Carol
2 Alice Bob	2 Alice Bob

Problem G. Panda Hunting Treasure Box

Source file name:	Panda.c, Panda.cpp, Panda.java, Panda.py
Input:	`Standard`
Output:	`Standard`

Our favorite treasure hunter, Panda, has been dropped in a two-dimensional maze with several treasure boxes. Each maze cell is either empty or has one treasure box in it. The treasure boxes contain money and no two boxes have the same amount, i.e., they are all distinct amounts. Panda can have only one of these boxes and, obviously, he would prefer the one with the highest amount but he may not have enough energy to get to the cell containing that box so Panda may have to settle for a lower amount.

Panda starts with some energy. The picture to the right shows the energy needed for Panda to move into any of the eight neighboring cells (note that the boundary cells in the maze have fewer than eight neighbors). To illustrate the energy needed for movements, if Panda wants to move up, it requires 2 units of energy; if he wants to move down, it requires 6 units of energy.

1	2	3
8	Panda	4
7	6	5

If Panda moves into a cell with a treasure box, that box is what he gets, i.e., the journey is over even if Panda still has some energy left to make more moves. And, obviously, Panda can make a move only if he has enough energy for that move. When Panda makes a move, his energy will go down accordingly to that move.

You are to determine the highest amount Panda can get. Note that Panda does not have to use all his energy, i.e., it is ok if Panda still has some energy left when he gets a treasure box.

Input

The first input line contains five integers:

- R_m ($2 \leq R_m \leq 500$), indicating the number of rows in the maze;
- C_m ($2 \leq C_m \leq 500$), indicating the number of columns in the maze;
- R_p ($1 \leq R_p \leq R_m$), indicating the row number of Panda's starting position;
- C_p ($1 \leq C_p \leq C_m$), indicating the column number of Panda's starting position; and
- E_p ($1 \leq E_p \leq 10^6$), indicating Panda's starting energy.

Assume that Panda's starting position is valid (i.e., it is in the maze) and that cell does not contain a treasure box.

Each of the next R_m input lines contains C_m integers (each integer between 0 and 10^6, inclusive). Each input line describes one row in the maze, providing the cell contents. A zero indicates there is no treasure box in that cell; non-zero indicates a treasure box (the amount in the box). Assume that there is at least one treasure box in the maze and that Panda has enough starting energy to reach one treasure box.

Output

Print the highest amount Panda can get.

Example

Input	Output
4 3 2 1 15 0 0 0 0 0 0 0 0 10 0 20 0	10
4 3 2 3 50 0 0 0 0 0 0 0 0 10 0 20 0	20
4 2 1 1 90 0 0 0 0 10 30 40 50	30

Problem H. Lecture Allocation

Source file name:	Lecture.c, Lecture.cpp, Lecture.java, Lecture.py
Input:	`Standard`
Output:	`Standard`

You are the coordinator for a competitive programming club. You need to hire some teachers to give lectures. There are a fixed number of lectures that need to be given this year. Additionally, there are a limited number of teachers that are willing to give lectures. Each teacher can teach up to three lectures, but not all the teachers need to teach a lecture, i.e., a teacher could teach 0, 1, 2, or 3 lectures. Each teacher charges a different amount depending on the number of lectures they give.

The money not spent will be used to fly the team to other contests, so you want to spend as little money as possible hiring enough teachers to give all the lectures.

Given the number of lectures to teach and how much each teacher charges for giving the lectures, determine the least amount of money necessary such that all the lectures will be taught.

Input

The first input line contains two integers, L and T ($1 \leq L \leq 5000$, $L/3 \leq T \leq L$), representing (respectively) the number of lectures and the number of teachers. Each of the following T input lines contains three integers, the i^{th} of which contains a_{i1}, a_{i2}, and a_{i3} ($0 < a_{i1} < a_{i2} < a_{i3} \leq 10^5$), representing (respectively) how much the i^{th} teacher charges to give 1, 2, and 3 lectures.

Output

Print on a single line by itself a single positive integer: the least cost for paying the teachers to cover all L lectures. Assume that there are enough teachers to cover all the lectures.

Example

Input	Output
4 3 8 10 20 10 20 30 11 17 25	27
6 2 10 20 25 30 35 37	62
5 2 10 20 25 30 35 37	57

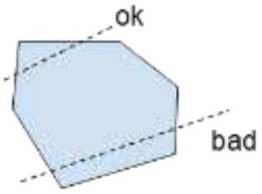
Explanation

For the first Example Input, the first teacher can give two lectures and the third teacher can give two lectures, so the total cost is $10 + 17 = 27$.

For the third Example Input, the first teacher can give two lectures and the second teacher can give three lectures, so the total cost is $20 + 37 = 57$.

Problem I. Stone Smoothing

Source file name:	Stone.c, Stone.cpp, Stone.java, Stone.py
Input:	`Standard`
Output:	`Standard`



You found a really sharp stone that could be described as a convex polygon. You decided to use some skills to make it smoother. You can "smooth" a corner (vertex) of the stone (polygon) by shaving it down some amount as long as you don't smooth down another corner at the same time, i.e., only one corner can be smoothed at a time.

Note that, when one vertex (corner) is smoothed, it is replaced by two new vertices. So, if desired, these new vertices (corners) can be smoothed.

To make the smoothest stone, you have to ensure that the largest exterior angle of the polygon is as small as possible, after smoothing a given stone a fixed number of times.

Geometry Refresher: An **exterior angle of a polygon** is an angle at a vertex of the polygon, outside the polygon, formed by one side and the extension of an adjacent side. In other words, if the side of a polygon is extended, **the angle formed outside the polygon** is the exterior angle.

Given the shape of the stone described using xy-coordinates and the number of times the stone could be smoothed, determine the smallest possible value for the largest resulting exterior angle.

Input

The first input line contains two integers, C and S ($1 \leq C \leq 5000$; $1 \leq S \leq 10^9$), representing (respectively) the number of corners in the stone and the number of times the stone can be smoothed. Each of the following C input lines contains two integers, the i^{th} of which are x_i and y_i ($-10^5 \leq x_i,\ y_i \leq 10^5$), representing the x coordinate and the y coordinate of the i^{th} corner of the stone (the vertices of the polygon are given in clockwise order).

Output

Print on a single line by itself a single floating-point number: the minimum possible value for the largest exterior angle after performing all the smoothing. Answers within 10^{-6} absolute of the expected answers will be considered correct.

Example

Input	Output
3 6 0 0 0 10 10 0	225.00000000000003
4 3 -1 1 1 1 1 -1 -1 -1	270.00000000000006

Problem J. Tic Tac Toe Counting

Source file name:	Counting.c, Counting.cpp, Counting.java, Counting.py
Input:	`Standard`
Output:	`Standard`

Tic Tac Toe is a simple children's game. It is played on a 3×3 grid. The first player places an `X` in any of the 9 cells. The next player places an `O` in any of the remaining 8 cells. The players continue to alternate placing `X`s and `O`s in unoccupied cells until either a player gets three of their symbols in a row, or the grid is full. If either player gets three in a row, in any of the three rows, three columns or two diagonals, that player wins and the game ends.

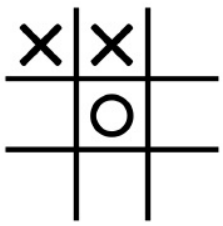
Fred and Sam are playing a game of Tic Tac Toe. In the middle of the game, Fred wonders: "How many games from this point in the game onward are winners for `X`? How many for `O`?" Two games are different if they have different sequences of moves, even if they result in the grid looking the same at any point.

Given a list of grids, determine first if they can be reached in a game of Tic Tac Toe, and if they can, how many games from that point on result in wins for `X`, and how many for `O`.

Input

The first line of input contains a single integer t ($1 \leq t \leq 10^5$), which is the number of test case grids.

Each of the next t lines contains a single string s ($|s| = 9$) which consists of only the characters '`.`', '`X`' and/or '`O`'. These are the test case grids. The first three characters represent the first row, the second three are the second row, and the last three are the last row, with '`.`' representing an empty cell. For example, the string `XX..O....` represents this grid:



Output

For each test case, output two space-separated integers. If the grid is not reachable in a game of Tic Tac Toe, output -1 -1. If the grid is reachable, output the number of games from that point on (including that grid) that are wins for `X`, followed by wins for `O`.

Example

Input	Output
4 XX..O.... X...OX... OOOX.X.X. OOOXXX...	191 194 232 200 0 1 -1 -1

Problem K. Contact Tracing

Source file name:	Tracing.c, Tracing.cpp, Tracing.java, Tracing.py
Input:	`Standard`
Output:	`Standard`

A novel infectious disease has started spreading through the population. You are tasked with figuring out who might be infected in order to get them to quarantine. The behavior of the disease among members of the population is as follows:

- When a person comes into contact with someone who is infectious, they may (but do not always) catch the disease. If they catch the disease, they are not infectious for the rest of that day, so they will not spread it to other people they come into contact with that day.
- When a person catches the disease, they are infectious starting the day after they caught the disease.
- Starting two days after they caught the disease, they are symptomatic; as a result, they will quarantine themselves and not come into contact with any other people from then on.

You know that on day 0, a single Patient Zero (of unknown identity) caught the disease. On day 1, they were infectious and potentially came into contact with people, potentially infecting those people. On day 2, Patient Zero became symptomatic and therefore stopped coming into contact with other people, but any people who were infected on day 1 could potentially spread the disease to the people they come into contact with.

Today is day k, and it is the end of the day. You have access to a list of all pairs of people who came into contact with each other on each day, including today. If you can identify all people who could be infectious but not symptomatic tomorrow (because they caught the disease today) and tell them to quarantine, you can halt the outbreak, because all people who are symptomatic tomorrow will quarantine themselves anyway. What is the minimum number of people you need to tell to quarantine to be sure that you halt the outbreak?

Input

The first line of input contains three integers n ($1 \leq n \leq 10^3$), k ($1 \leq k \leq 10$), and c ($0 \leq c \leq 10^3$), where n is the number of people, k is the number of days since Patient Zero became infected, and c is the number of contacts that occurred between people.

Each of the next c lines contains three space-separated integers a, b ($1 \leq a < b \leq n$), and d ($1 \leq d \leq k$), denoting that people a and b came into contact with each other on day d. All of these c lines will be distinct. There will be at least one person with no contacts after day 1.

Output

Output a line with a single integer x denoting the minimum number of people you must tell to quarantine beginning on day $k + 1$. Then, output x lines listing the people who must quarantine, one per line, in ascending order.

Example

Input	Output
4 2 4 1 4 1 2 3 2 2 4 1 3 4 1	2 2 3

Explanation

In Example Input 1, Patient Zero could be person 1 or person 4. Persons 2 and 3 can be eliminated, because they had contact on day 2, when Patient Zero would be symptomatic and quarantined.

Suppose person 1 is Patient Zero. Person 1 may have infected person 4 on day 1. Person 1 is symptomatic on day 2, so they begin quarantining. Therefore, there is no chain of contact from person 1 (Patient Zero) to persons 2 or 3, so person 2 and person 3 are safe and do not need to be quarantined. If person 1 infected person 4 on day 1, then person 4 will be symptomatic tomorrow (day 3), so there is no need to contact them because they will isolate on their own. Therefore if Patient Zero is person 1, no one needs to be notified.

Now suppose person 4 is Patient Zero. Person 4 may have infected person 1 on day 1; using the same logic as above, there is no need to notify person 1 or person 4 to quarantine on day 3. This leaves persons 2 and 3, which requires us to consider multiple infection patterns.

If person 4 infected both persons 2 and 3 on day 1, then both of them will be symptomatic and self-quarantine starting on day 3, so we do not need to notify them.

Suppose instead person 4 infected person 2 but NOT person 3 on day 1. (Remember that transmission is not guaranteed.) In this case, person 2 could go on to infect person 3 on day 2, and person 3 would not yet be symptomatic on day 3, so we would need to notify person 3 to quarantine.

Similarly, if person 4 infected person 3 but NOT person 2 on day 1, we would need to notify person 2 to quarantine.

Therefore to be sure we can halt the outbreak, we need to notify both person 2 and person 3 to quarantine. This is guaranteed to stop the outbreak, whether person 1 or person 4 is Patient Zero.

Problem L. Triangular Logs

Source file name:	Triangular.c, Triangular.cpp, Triangular.java, Triangular.py
Input:	`Standard`
Output:	`Standard`

The local forest has a lot of trees! Each tree is located at integer coordinates and has an integer height. Cutting down any tree gives you a log with a length equal to its height. You want to obtain three triangular logs (that is, three logs that form a non-degenerate triangle) by cutting down three trees.

Given a list of queries which each specify an axis-aligned rectangular region, can you obtain three triangular logs by cutting down three trees in that region, possibly including those on the boundary of the rectangle?

Input

The first line of input contains two integers n and q ($1 \leq n,\ q \leq 10^5$), where n is the number of trees and q is the number of queries.

Each of the next n lines contains three integers x, y and h ($1 \leq x,\ y,\ h \leq 10^9$), which describes a tree at location $(x,\ y)$ with height h. All tree locations are distinct.

Each of the next q lines contains four integers x_{low}, y_{low}, x_{high} and y_{high} ($1 \leq x_{low} \leq x_{high} \leq 10^9$, $1 \leq y_{low} \leq y_{high} \leq 10^9$), describing an axis-aligned rectangular region for a query.

Output

Output q lines. Each line contains a single integer, which is the answer to the given query. Output 1 if there are three trees in the queried region that can form a non-degenerate triangle, and 0 otherwise. Output answers to the queries in the order of the input.

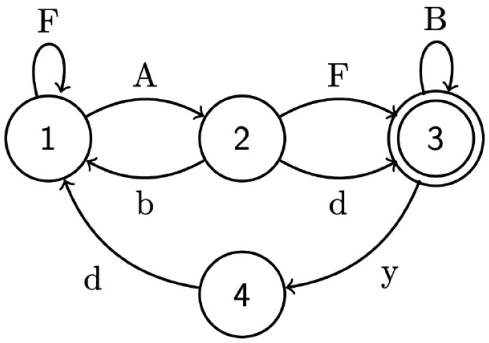
Example

Input	Output
9 5 1 3 3 2 3 1 3 3 4 1 2 1 2 2 5 3 2 9 1 1 2 2 1 6 3 1 5 1 1 1 2 1 1 2 2 1 1 1 3 1 2 3 2 1 1 3 3	0 1 0 0 1

Problem M. Transparency

Source file name:	Transparency.c, Transparency.cpp, Transparency.java, Transparency.py
Input:	`Standard`
Output:	`Standard`

You must audit a factory that produces a number of products. Each product is subject to its own taxation rules and regulations. Unfortunately, deciding which rules apply to a product is not obvious since it can be difficult to distinguish the products from one another. It is your job to investigate if the factory is being completely *transparent* in its production process by determining if the factory is capable of producing two different products that are indistinguishable from one another.



The factory design from the first example input.
The packaging station is marked by a double circle.

The final products are represented by sequences of uppercase letters (‘`A`’-‘`Z`’) and lowercase letters (‘`a`’-‘`z`’). Two final products are indistinguishable from one another if the products are equal after removing all lowercase letters. For example, `WabXYcz` is indistinguishable from `gWXfbY`, since after removing all lowercase letters we are left with `WXY` in both instances.

A factory is modelled as a set of stations S, a set of directed and labelled transition edges T, an initial station s_0, and a set of packaging stations P. The initial station is in the set of stations, i.e., $s_0 \in S$, and the set of packaging stations is a subset of S, i.e., $P \subseteq S$. A transition edge is defined by a 3-tuple $(s_1,\ \sigma,\ s_2)$, where $s_1,\ s_2 \in S$ are stations, and σ is an uppercase or lowercase letter. The existence of the transition $(s_1,\ \sigma,\ s_2)$ in T means that if in producing some product we are at station s_1, then appending σ to the product will leave us at station s_2. That is, there is a directed edge from station s_1 to station s_2, labelled σ.

A product can be produced by the factory if by starting at station s_0 there is a sequence of edges that can be followed, ending at a station in P, whose edge labels can be concatenated, in order, to create the product. The sequence of edges can be empty if $s_0 \in P$. For example, the following strings can all be produced by the factory described in the first example input and illustrated in the figure: `AF`, `FAFB`, `AbFFAd`, `AdydAd`. Note that this is not an exhaustive list.

Each production transition can be used an unlimited number of times to make a product. It is guaranteed that for each station, s, and letter, σ, there is at most one directed edge from station s labelled σ. That is, $(s_1,\ \sigma,\ s_2)$, $(s_1,\ \sigma,\ s_3) \in T$ implies $s_2 = s_3$. It is possible that transitions will go back to the same station; that is, $(s_1,\ \sigma,\ s_1) \in T$ is allowed.

Given a factory design, determine if the factory is capable of producing two distinct but indistinguishable products. If there is such a pair of products, then report the sum of the lengths of the strings in the pair that minimizes the sum of the lengths. If there is no such pair, print -1.

Input

The first line of input contains three integers, s ($1 \le s \le 50$), p ($1 \le p \le s$) and t ($1 \le t \le 52 \cdot s$), where s is the number of stations, p is the number of packaging stations, and t is the number of transitions. The set of stations is numbered from 1 to s.

Each of the next p lines contains a single integer i ($1 \le i \le s$). These are the packing stations. All values of i are distinct.

Each of the next t lines is of the form:

$s_1\ \sigma\ s_2$

where s_1 and s_2 ($1 \leq s_1,\ s_2 \leq s$) are stations, and σ is a single character, consisting of an uppercase or lowercase letter. These are the transitions. The initial station is always station 1. There will never be two transitions out of the same state labelled with the same character.

Output

Output a single line with a single integer. If there is no pair of distinct, indistinguishable products, then output -1. If there exists a pair of distinct, indistinguishable products, then output the smallest sum $|a| + |b|$ where a and b are strings corresponding to distinct, indistinguishable products.

Example

Input	Output
4 1 8 3 1 F 1 1 A 2 2 b 1 2 F 3 2 d 3 3 B 3 3 y 4 4 d 1	10
4 1 8 3 1 F 1 1 A 2 2 b 1 2 F 3 2 d 3 3 B 3 3 y 4 4 d 4	-1
5 2 5 1 5 1 i 2 2 c 3 3 p 4 4 c 1 1 I 5	4