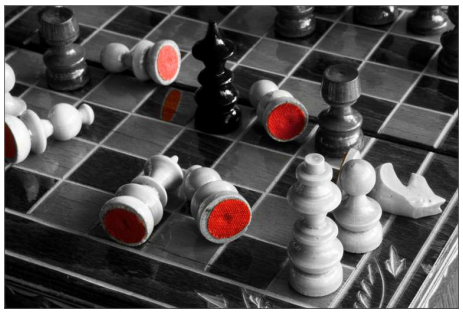
Problem A. Assassins

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

In the cut-throat world of assassins for hire, the rivalry is ruthless and everyone is fighting to get an edge. To eliminate the competition, many assassins even go so far as to assassinate other assassins. The question is: with several assassins trying to do each other in, which ones will remain alive and kicking, and which ones will kick the bucket?



Picture by Cristian V. from flickr, cc by-nd.

Assassins generally lay careful plans before executing them, including planning multiple attempts to dispose of the same target, with the second attempt being a backup in case the first attempt fails, the third attempt being a secondary backup, and so on. Using their great annihilytical skills, assassins can also very accurately determine the probability that any given assassination attempt will succeed.

Given the list of planned assassination attempts for a group of assassins, what are the probabilities that each assassin is alive after all these attempts? Performing an assassination attempt requires that the assassin is still alive, so if the assassin is indisposed due to already having been assassinated, the attempt is cancelled.

Input

The first line of input contains two integers n and m, where n ($1 \leq n \leq 15$) is the number of assassins, and m ($0 \leq m \leq 1000$) is the number of planned assassination attempts. The assassins are numbered from 1 to n.

Then follow m lines, each containing two integers i, j, and a real number p, indicating that assassin i plans to attempt to assassinate assassin j ($1 \leq i, j \leq n$, $j \neq i$), and that this attempt would succeed with probability p ($0 \leq p \leq 1$, at most 6 digits after the decimal point). The planned attempts are listed in chronological order from first to last, and no two attempts happen simultaneously.

Output

Output n lines, with the i'th containing the probability that assassin i is alive after all m assassination attempts have taken place. You may assume that none of the n assassins will die of any other cause than being assassinated in one of these m attempts. The probabilities should be accurate to an absolute error of at most 10^{-6}.

Example

Input	Output
4 3 1 2 0.25 1 4 0.42 2 3 1.0	1 0.75 0.25 0.58
2 3 1 2 0.23 2 1 0.99 1 2 0.99	0.2377000000 0.7623770000

Problem B. Brexit Negotiations

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

As we all know, Brexit negotiations are on their way—but we still do not know whether they will actually finish in time.

The negotiations will take place topic-by-topic. To organise the negotiations in the most effective way, the topics will all be discussed and finalised in separate meetings, one meeting at a time.

This system exists partly because there are (non-cyclic) dependencies between some topics: for example, one cannot have a meaningful talk about tariffs before deciding upon the customs union. The EU can decide on any order in which to negotiate the topics, as long as the mentioned dependencies are respected and all topics are covered.

Each of the topics will be discussed at length using every available piece of data, including key results from past meetings. At the start of each meeting, the delegates will take one extra minute for each of the meetings that has already happened by that point, even unrelated ones, to recap the discussions and understand how their conclusions were reached. See Figure 2 for an example.

Nobody likes long meetings. The EU would like you to help order the meetings in a way such that the longest meeting takes as little time as possible.

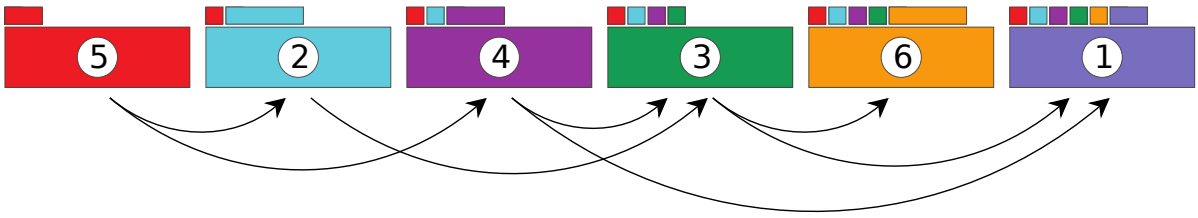


Illustration of how time is spent in each meeting in a solution to Sample Input 2.

Input

The input consists of:

- One line containing an integer n ($1 \leq n \leq 4 \cdot 10^5$), the number of topics to be discussed. The topics are numbered from 1 to n.
- n lines, describing the negotiation topics.

  The ith such line starts with two integers e_i and d_i ($1 \leq e_i \leq 10^6$, $0 \leq d_i < n$), the number of minutes needed to reach a conclusion on topic i and the number of other specific topics that must be dealt with before topic i can be discussed.

  The remainder of the line has d_i distinct integers $b_{i,1}, \ldots, b_{i,d_i}$ ($1 \leq b_{i,j} \leq n$ and $b_{i,j} \neq i$ for each j), the list of topics that need to be completed before topic i.

It is guaranteed that there are no cycles in the topic dependencies, and that the sum of d_i over all topics is at most $4 \cdot 10^5$.

Output

Output the minimum possible length of the longest of all meetings, if meetings are arranged optimally according to the above rules.

Example

Input	Output
3 10 0 10 0 10 0	12
6 2 2 4 3 4 1 5 1 2 2 4 3 1 5 2 0 4 1 3	8

Problem C. Cars

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

You work as an engineer at the leading self-driving car company Wayber. Wayber just got approval from the government of Sweden for their cars to drive in the streets of Stockholm. Unfortunately, all your tests so far have been with one car at a time on a closed dirt track. You are not really prepared for what will happen when two of the cars encounter each other in the city, and want to write some code to handle this.

Picture by Sammmerbatter via Wikimedia Commons, cc by-sa

The cars are already good at staying inside the lines and turning. However, they are not very good at detecting other moving vehicles. Fortunately, the government of Sweden is so enthusiastic about self-driving cars that they have banned all other forms of transportation in Stockholm, including human-driven cars, biking, and walking. If you can just detect whether two cars will collide you will be able to build a safe system.

You know that all cities consist of only north-south and east-west oriented streets in a perfect grid, and all cars are perfect rectangular prisms. When detecting collisions you only need to worry about cars traveling at a constant speed without turning.

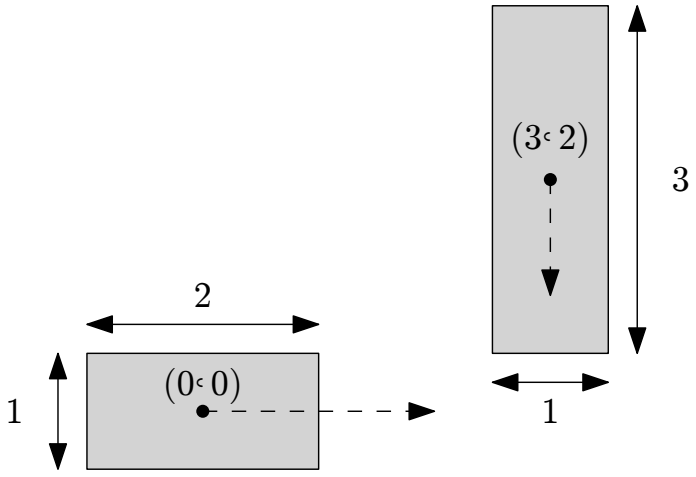


Illustration of Sample Input 1.

Input

The first line of input contains an integer $1 \leq t \leq 10^4$, the duration in seconds of the trajectory of the two cars. Then follow the description of the trajectory of two cars.

A trajectory consists of a line containing a character d and five integers x, y, s, w and l. The starting position of the car is (x, y) $(0 \leq x, y \leq 10^4)$ and its direction is d which is either `N` (positive y direction), `S` (negative y direction), `W` (negative x direction) or `E` (positive x direction). The car is travelling at a speed of $1 \leq s \leq 10^4$ units per second, is $1 \leq w \leq 10^4$ units wide, and $1 \leq l \leq 10^4$ units long.

Cars start out centred on their starting coordinates and do not initially overlap in a nonzero area.

Output

For each line of input, output a line "`crash`" if the two cars will crash or "`safe`" if they will not crash. If two cars overlap in an area of size zero (only on an edge or corner) they do not crash.

Example

Input	Output
5 E 0 0 2 1 2 S 3 2 1 1 3	crash
1 E 0 0 1 1 1 N 0 2 1 1 1	safe
2 N 0 0 7 3 1 S 3 20 12 4 1	crash
1 N 0 0 7 3 1 S 3 20 12 4 1	safe

Problem D. Date Pickup

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

Richard and Janet are going on their first date. Richard has offered to meet her at home with his bicycle, and Janet tells him she will call when she is ready in 10 to 20 minutes. But Richard is an impatient person; while he could wait at home for Janet's signal, he might also leave early and travel around the neighbourhood for a bit, in order to minimise the time it takes him to reach her once she calls. Due to his impatience, once Richard is on his bicycle, he does not want to ride any slower than the legal speed limit, stop at intersections, or wait outside Janet's house (but he does not mind passing by Janet's house and returning to it later).

Given the directed graph representing the neighbourhood around Richard's and Janet's houses, Richard wants to devise a route around the neighbourhood (after an optional waiting period at his own house) which minimises the time that Janet has to wait in the worst case. He can travel for as long as he likes and visit each intersection as many times as he likes.

Janet will call Richard as soon as she is ready, and at that point Richard will take the shortest path to her that he can. Richard does not know exactly when Janet will be ready, but he knows it will be in somewhere between a and b minutes (not necessarily at a whole minute).

If Richard is passing through an intersection at the exact same instant Janet calls, the call is considered to happen before he chooses what to do at the intersection. For example, if he is passing by Janet's house at the moment she calls, he can immediately stop there and she does not have to wait for him at all.

It could happen that Janet never has to wait for w minutes, but that she might have to wait for $w - \epsilon$ minutes for arbitrarily small $\epsilon > 0$, if she calls Richard at some inopportune moment (say, nanoseconds after he has left an intersection). In this case, we still define the worst case waiting time to be w.

Input

The input consists of:

- One line with two integers a, b ($0 \leq a \leq b \leq 10^{12}$), indicating that Janet will be ready in at least a minutes and at most b minutes.
- One line with two integers n, m ($2 \leq n \leq m \leq 10^5$), the number of intersections and the number of roads in the neighbourhood. The intersections are numbered from 1 to n.
- m lines, each with three integers u, v and t ($1 \leq u, v \leq n$, $1 \leq t \leq 10^6$), indicating that there is a one-way road from intersection u to intersection v, and that it takes Richard exactly t minutes to travel along this road.

Richard's house is at intersection 1 and Janet's house is at intersection n. It is guaranteed that it is possible to travel from Richard's house to Janet's house, and that it is possible to exit each intersection through at least one road, even if that road just loops back to the same intersection.

Output

Output the time Janet has to wait in the worst case assuming she will be ready in at least a minutes and at most b minutes and Richard plans his route optimally.

It can be shown that the worst case waiting time is always an integer.

Example

Input	Output
10 20 3 5 1 3 7 2 1 1 2 3 2 2 3 5 3 2 4	6
4 10 5 7 1 4 6 4 5 5 4 5 3 5 5 30 1 2 1 2 3 1 3 2 1	5

Problem E. Equality Control

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

In programming contest circles, one of the most important roles is that of the Chief Equality Officer (CEO). This person is responsible for making sure that every team has an equal chance of winning the contest. Since last year's NWERC the current CEO, Gregor, has thought at length about how to make the contest even more fair and equal.

His answer is to introduce a new programming language as the only one allowed for submissions. This way, no team will be disadvantaged by not having mastered any of the allowed languages. This language is called BALLOON, short for Building A Long List Of Ordinary Numbers. Its only data type is the list of integers. To keep the language fast, it contains only four instructions:

- `[`x_1`,...,`x_n`]` is the constructor for lists. It returns the integers inside the brackets in their given order.
- `concat(<Expr`$_1$`>,<Expr`$_2$`>)` returns a list of all the integers returned when evaluating the expression `<Expr`$_1$`>` followed by all of the integers returned when evaluating `<Expr`$_2$`>`.
- `shuffle(<Expr>)` returns a list of all the integers returned by `<Expr>`, rearranged according to a uniformly random permutation, i.e., each permutation of the elements is used with equal probability.
- `sorted(<Expr>)` returns a list of all the integers returned by `<Expr>`, rearranged into non-decreasing order.

As an example, consider the first expression of Sample Input 1. The two shuffle exressions both take the list `[1,2]` as input and return one of the lists `[1,2]` and `[2,1]`, each with probability 0.5 (independently of each other). The outer concat operator takes the two returned lists as its input and returns their concatenation. I.e., it returns one of the lists `[1,2,1,2]`, `[1,2,2,1]`, `[2,1,1,2]`, and `[2,1,2,1]`, each with probability 0.25.

Naturally, we cannot use byte-by-byte output comparison any more when teams submit their solutions in BALLOON, as its output is probabilistic. The judge server instead has to check whether a submitted program is equivalent to the sample solution created by the judges. Two programs are equivalent if for any list L of integers, both programs have an equal probability of returning L.

It is your task to determine whether two given BALLOON programs are equivalent.

Input

The input consists of:

- One line containing a string `A`, the first program.
- One line containing a string `B`, the second program.

Each program is a syntactically valid BALLOON program with between 3 and 10^6 characters, and contains neither spacing nor empty lists (i.e., the strings " " or "`[]`" do not occur in the input).

Each integer in each program is greater than 0 and less than 10^9.

Output

If the two programs are equivalent, output "`equal`", otherwise output "`not equal`".

Example

Input
concat(shuffle([1,2]),shuffle([1,2])) shuffle([1,2,1,2])
Output
not equal

Input
sorted(concat([3,2,1],[4,5,6])) [1,2,3,4,5,6]
Output
equal

Input
concat(sorted([4,3,2,1]),shuffle([1])) concat(concat([1,2,3],shuffle([4])),sorted([1]))
Output
equal

Problem F. Final Standings

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

The GCPC 2019 is finally over. You have worked for five hours and have solved as many problems as possible. However, you still do not know which place you have got, since the scoreboard is still frozen. Of course you think that you and your team have won GCPC 2019.

As there is always a lengthy pause between the end of the contest and unfreezing the scoreboards (someone has to print the certificates and someone has to re-compile the solution slides), you want to use your time to determine how probable it is that your team has won GCPC 2019.

You know the final scoreboard, that is, for all teams which problems they have solved prior to the freeze and which problems they have attempted during the freeze. Also you know from last year how good each team is and you trust your own estimates of each problem's difficulty. To be precise, if a team with strength $s \in [0, 1]$ has attempted to solve a problem with difficulty $d \in [0, 1]$, you assume that the probability of solving the problem is $s \cdot d$.

As your team was very quick in solving the simple problems this year, you assume that a team can only have a higher place than you, if they have solved more problems (you assume that you will always win the tie-break).

Input

- The first line of input contains two integers t and p ($1 \leq t, p \leq 100$), the number of teams and the number of problems in the contest. The teams are numbered 1 to t and the problems are numbered 1 to p. Your team is team t.
- The second line contains $t - 1$ real numbers $s_1, \ldots, s_{t-1}$ ($0 \leq s_i \leq 1$ for each i), where s_i is the strength of team i.
- The third line contains p real numbers $d_1, \ldots, d_p$ ($0 \leq d_j \leq 1$ for each j), where d_j is the difficulty of problem j.
- Then follow $t - 1$ lines describing the state of the problems for the other teams. Every such line contains p characters $c_1, \ldots, c_p$, where c_j in the ith line describes the state of problem j for team i and is `X` if the team has solved the problem before the freeze, `?` if the team submitted a program for this problem after the freeze, and `-` otherwise.
- The last line of input describes the problems your team solved with p characters, each being either `X` or `-`.

All real numbers in the input have at most six decimals.

Output

Output a single real number, the probability that your team won GCPC 2019. Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
3 3 0.95 0.95 0.95 0.95 0.95 ? ? ? X X - X X -	0.264908
2 5 0.5 0.1 0.2 0.3 0.4 0.5 ? ? ? ? ? X - - - -	0.8387625

Problem G. Global Warming

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

This is a very exciting week for John. The reason is that, as a middle school teacher, he has been asked to dedicate the entire week to teaching his class of n students about the cause and effect of global warming. As John is very passionate about his planet, he's going to spend extra time and effort to make this week memorable and rewarding for the students. Towards that, one of the things he wants to ask them to do is to prepare, as homework, presentations about global warming. To make this a little easier for them, as well as more fun, he has asked them to do this in groups of two.

Of course arranging the students into groups comes with the usual headache, namely that only friends are willing to work together. Luckily the students in his class are a friendly bunch. In particular, if p, q and r are three distinct students, and p and q are friends, and q and r are friends, then p and r are also friends. But John now realizes the irony in asking his students to work at home in groups, as students may have to travel to meet their group partner, which may emit greenhouse gases such as carbon dioxide, depending on their mode of transportation. In the spirit of this week's topic, John asked all the students in his class to calculate, for each of their friends, how much carbon dioxide would be emitted if they were to meet up with the respective friend.

Using this information, can you help John figure out what is the minimum total amount of carbon dioxide that will be emitted if he arranges the groups optimally, or determine that it's not possible to arrange all the students into groups of two friends?

Input

The first line contains two integers n and m ($1 \leq n \leq 200$, $0 \leq m \leq 250$), the number of students in John's class, and the total number of pairs of friends in the class. As John is bad with names, he has given each of his students a distinct integer identifier between 1 and n.

Each of the next m lines contains three integers p, q and c ($1 \leq p, q \leq n$, $0 \leq c \leq 10^6$), the identifiers of two distinct students that are friends, and how many grams of carbon dioxide would be emitted if they were in a group together, and thus had to meet. Each pair of friends is listed exactly once in the input.

Output

Output the minimum total amount of carbon dioxide, in grams, that would be emitted if John arranges all students optimally into groups of two friends, or "`impossible`" if there is no way to arrange the students into groups in that way.

Example

Input	Output
5 4 3 1 375 2 5 283 1 4 716 3 4 98	impossible
6 7 5 6 600 2 5 200 3 5 400 6 3 500 1 4 300 3 2 400 6 2 200	900

Problem H. Ski Lifts

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

Last winter, an avalanche swept away all the ski lifts from the ski resort Valen. Instead of rebuilding the lifts like they were before, the plan is to do it in a more optimized way, and you are responsible for this.

Picture by SkiHoodoo from Wikimedia Commons, cc by-sa

The only thing remaining from the old lift system are n pylons situated at integer coordinates in the plane. You would like to put lifts in the form of line segments between some of these pylons. The line segments must satisfy the following constraints:

1. A line segment can only go between pylons (x_1, y_1) and (x_2, y_2) if $|y_1 - y_2| = 1$.
2. There are two types of pylons, one-way and two-way pylons. The one-way pylons can be connected to at most one other pylon, and the two-way pylons can be connected to at most two other pylons. However, if a two-way pylon i is connected to two other pylons, then they must be on opposite sides of i in the y-direction. In other words, the two pylons connected to i must have different y-coordinates.
3. Two line segments may not intersect (except that the two line segments incident on a two-way pylon may touch at their endpoints).

What is the maximum number of ski lifts (line segments) you can place under these constraints?

Input

The first line contains one integer n ($1 \leq n \leq 10^5$). Each of the following n lines contains three integers x, y, and a, the coordinates and type of a pylon ($0 \leq x, y \leq 10^5$; $a = 1$ for a one-way pylon and $a = 2$ for a two-way pylon). All the pylons are situated at different coordinates.

Output

Output the maximum number of ski lift line segments that can be placed.

Example

Input	Output
8 1 0 1 3 0 2 0 1 1 2 1 2 4 1 2 1 2 2 2 3 1 4 3 1	4
4 0 0 1 100000 1 1 0 99999 1 100000 100000 1	2

Problem I. Inflation

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

For NWERC 2018, the organisers have done something rather special with the balloons. Instead of buying balloons of equal size, they bought one balloon of every integer size from 1 up to n. A balloon of size s has a capacity of s decilitres.

To avoid inflating the balloons by hand, the organisers also bought n helium gas canisters. Each canister can only be used to inflate one balloon, and must be emptied completely into that balloon (it is not possible to disconnect a canister from a balloon before the canister has been fully used).

Unfortunately the gas canisters were bought at a garage sale, and may contain differing amounts of helium. Some may even be empty! To make the best of this challenging situation, the canisters will have to be paired with the balloons smartly.

The organisers want to assign all of the gas canisters to separate balloons, such that the balloon that is inflated the least (relative to its capacity) still contains the maximum possible fraction of helium inside. What is the maximum such (minimum) fraction that is possible?

Balloons filled beyond their capacity will explode. Explosions are upsetting and must be avoided.

Input

The input consists of:

- One line with the integer n ($1 \leq n \leq 2 \cdot 10^5$), the number of balloons and gas canisters.
- One line with n integers $c_1, \ldots, c_n$ ($0 \leq c_i \leq n$ for each i), the amounts of helium in the gas canisters, in decilitres.

Output

If it is possible to fill all the balloons without any exploding, output the maximum fraction f such that every balloon can be filled to at least f of its capacity. Otherwise, output "`impossible`".

Your answer should have an absolute error of at most 10^{-6}.

Example

Input	Output
6 6 1 3 2 2 3	0.6
2 2 2	impossible
5 4 0 2 1 2	0

Problem J. Jinxed Betting

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

Julia is betting on a large sporting competition involving matches between pairs of teams. There are no parallel matches and each bettor receives one point for every correct bet they make. Julia had a good streak and is in the lead. Now she worries that her good luck may be turning, and decides to change her strategy.

She collaborates with a betting shop owner who tells her the bets made by everyone else. Whenever Julia makes a bet, she first checks the bets of all bettors with the most points so far (except herself of course) and then chooses the same team as the majority. In the case of a tie, she bets on her favourite of the two teams in the game.

Julia wants to know for how many more matches she is guaranteed to stay in the lead in the worst case (i.e., no matter what bets the others make or what the outcomes of the games are). For this problem we consider Julia to be in the lead if there is no other bettor that has strictly more points than her.

For example, suppose as in Sample Input 1 that Julia initially has three points, and there are two other bettors with three and two points respectively. For the first match, she will make the same bet as the other bettor with three points. If this person bets on the losing team and the other bets on the winning team all players have an equal score of three points. If for the next match the other two persons bet on different teams, Julia places the bet on her favourite, which of course may lose. Thus after two more matches she might lose the lead.

Input

The input consists of:

- One line with an integer n ($3 \leq n \leq 10^5$), the number of people who place their bets.
- One line with n integers $p_1, \ldots, p_n$ ($0 \leq p_i \leq 10^{16}$ for each i), the points of all people who play the betting game. The first of these numbers corresponds to the score of Julia. You may assume that no other score exceeds Julia's score in the beginning.

Output

Output the number of matches for which Julia is guaranteed to stay in the lead.

Example

Input	Output
3 3 3 2	1
5 8 4 3 5 2	6

Problem K. Kleptography

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

John likes simple ciphers. He had been using the "Caesar" cipher to encrypt his diary until recently, when he learned a hard lesson about its strength by catching his sister Mary browsing through the diary without any problems.

Rapidly searching for an alternative, John found a solution: the famous "Autokey" cipher. He uses a version that takes the 26 lower-case letters '`a`'–'`z`' and internally translates them in alphabetical order to the numbers 0 to 25.

The encryption key k begins with a secret prefix of n letters. Each of the remaining letters of the key is copied from the letters of the plaintext a, so that $k_{n+i} = a_i$ for $i \geq 1$. Encryption of the plaintext a to the ciphertext b follows the formula $b_i = a_i + k_i \bmod 26$.

Mary is not easily discouraged. She was able to get a peek at the last n letters John typed into his diary on the family computer before he noticed her, quickly encrypted the text document with a click, and left. This could be her chance.

Input

The input consists of:

- One line with two integers n and m ($1 \leq n \leq 30$, $n + 1 \leq m \leq 100$), where n is the length of the keyword as well as the number of letters Mary saw, and m is the length of the text.
- One line with n lower-case letters, the last n letters of the plaintext.
- One line with m lower-case letters, the whole ciphertext.

Output

Output the plaintext of John's diary.

Example

Input	Output
5 16 again pirpumsemoystoal	marywasnosyagain
1 12 d fzvfkdocukfu	shortkeyword

Problem L. Lights Out

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

Poor Eve, she is almost always the last one at work. Most unfortunately, she is very afraid of the dark, and the company rules dictates the last one leaving the office is obliged to make sure all the lamps at the whole office are off. Her sloppy colleagues sometimes forget to turn their lamps off, more often then not to be honest, but to be fair it is not as trivial as you might think. You see, at Eve's workplace, each room has exactly one lamp but may have several light switches. The thing is though, that unlike a traditional light switch, these switches toggle a subset of all the lamps at the office. Each switch inverts the light status, either from lit to turned off or the other way around, for a fixed set of lamps depending on the switch. It may even be that the lamp in the room is not effected by any of its switches, or that there are no switches in the room at all.

Picture by Comfreak on pixabay, cc0

Eve is a creature of habit and wants to take a fixed route out of the office each evening. At the same time, the set of lamps to turn off may be different on different days, so she has to plan for all eventualities. In other words, Eve wants a route through the office such that, for any possible configuration of the lamps, there is some combination of the light switches in the rooms Eve moves through that will allow her to turn off that configuration of lamps.

Note that it may be the case that some configurations of lamps are impossible to turn off (for instance, in Sample Input 1, it would be impossible to turn off only lamp 0), but Eve doesn't have to worry about such configurations of lamps (because they are also impossible to turn on). The route only has to let her turn off configurations which are actually possible to turn off. Eve is fine with passing through a room even if its lamp is turned off, for instance it is OK if the last lamp in the office is turned off somewhere in the middle of the route even though it means walking through a few unlit rooms at the end.

Input

The first line of input contains three positive integers, n, m, and l ($2 \leq n \leq 20$, $1 \leq m \leq 190$, $1 \leq l \leq 100$), where n is the number of rooms in the office, m is the number of connections between rooms, and l is the number of switches. Next follow m lines, each describing a pair of adjacent rooms containing two room numbers $a \neq b$, meaning you can enter room b from room a and vice versa. No unordered pair $\{a, b\}$ appears more than once.

Then follow l lines, each describing a light switch. Each such line starts with a room number telling which room the light switch is in. The second integer on the line $p > 0$ gives the number of lamps that are toggled by the switch. The remainder of the line contains p room numbers. You can assume that no two room identifiers are identical in a switch's toggle list.

The rooms are numbered 0 through $n - 1$. The room from which Eve leave's the office is number 0, and Eve's room (where she starts) is number 1. You may assume that it is possible to reach any room from room 1.

Output

Output one line with the minimum number of rooms on a path from Eve's room to the entrance room (counting both endpoints) including a set of switches making it possible to turn off any possible subset of lamps lit. Note that she might need to visit a room multiple times, in which case each visit should be counted.

Example

Input	Output
6 5 7 0 2 2 3 3 4 3 5 1 2 1 2 0 1 1 2 1 2 2 2 2 3 3 2 3 4 4 2 4 5 4 2 0 5 5 2 0 3	7