Problem A. Beer Barrels

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

Finally, you got into the cellar of your preferred brewery where you expected many large piles of beer barrels to be stored. You are eager to inspect the barrels and maybe even their content (a lot and lot of content, actually...). Unfortunately, you find only five barrels, all hopelessly empty and dry. Some numbers are painted on the first four barrels, one number on each of them. A note is attached to the fifth barrel. Behind the barrels in the dark, there is some low and barely discernible door in the wall, leading quite obviously to another lower cellar where you hope a whole slew of full barrels is kept hidden. The door is locked with a heavy and complex looking lock. With no obvious further constructive action in mind, you sit down to study the note on the fifth barrel.

Its essence is the following.

Denote the numbers painted on the first, second, third and fourth barrel by A, B, K and C. Numbers A, B and C are just single digits.

Now imagine that in the distant future some incredibly powerful computer (powered by quantum yeast) prints a list of all numbers which have exactly K digits and in which each digit is equal to A or B. Another equally mighty computer then takes the list and also the value C as the input and calculates the number of occurrences of digit C in the whole list.

The resulting number taken modulo $10^9 + 7$ is to be typed into the door lock to open it and to gain access to the lower cellar.

You decide to calculate that number in your notebook you took with you.

Input

The input consists of a single line with four integers A, B, K, C ($1 \leq A, B, C \leq 9$, $0 \leq K \leq 1000$) which represent the numbers painted on the first four barrels.

Output

Output a single integer which opens the door lock.

Example

Input	Output
1 2 3 2	12
6 5 2 6	4

Problem B. Beer Bill

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

Pub bills were introduced into daily life long before computers even existed. People tend to think that once a bill has been paid, it is no more than a pathetic paper scrap not worthy of any more attention. The fact, completely overlooked by established computer science streams, is that the scribblings on the bill represent highly formalized and often quite nontrivial text suitable for many formal language applications and analyses.

A bill consists of lines of characters. Each line is either a priced line or a rake line. A priced line begins with a positive integer — the price of a food or drink item — which is optionally followed by some number of vertical bars. The price of the line is calculated as follows: If the bars are present, the price of the line is equal to the item price multiplied by the number of bars. Otherwise, the price of the line is equal to the price of the item. A rake line contains only vertical bars (similar to rake dents, hence the name rake line), each bar stands for one beer bought by the bill holder. The price of the rake line is the price of one beer multiplied by the number of the bars on the line. The beer price, in this problem, is equal to 42 monetary units. The bill total is the total of prices of all lines on the bill.

We present you with a formal definition of the so-called Raked bill language, as far as we know, the first of its kind in the whole history of computer science. The bills in this problem are expressed in the Raked bill language.

```
<BILL>          ::= <LINE> | <LINE><BILL>
<LINE>          ::= <PRICED_LINE><line_break> | <RAKE_LINE><line_break>
<PRICED_LINE>   ::= <PRICE_SPEC> | <PRICE_SPEC><RAKE>
<RAKE_LINE>     ::= <RAKE>
<PRICE_SPEC>    ::= <PUB_INTEGER><comma><hyphen>
<RAKE>          ::= <rake_dent> | <rake_dent><RAKE>
<PUB_INTEGER>   ::= <dig_1_9> | <dig_1_9><DIG_SEQ>
<DIG_SEQ>       ::= <dig_0_9> | <dig_0_9><DIG_SEQ>
<dig_1_9>       ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<dig_0_9>       ::= '0' | <dig_1_9>
<rake_dent>     ::= '|'         // ascii character 124
<comma>         ::= ','         // ascii character 44
<hyphen>        ::= '-'         // ascii character 45
<line_break>    ::= LF          // ascii character 10, line feed
```

In the language specification above, the actual characters which appear on the bill are enclosed in single quotation marks to distinguish them from the other parts of the specification. The symbol `//` introduces a single line comment, it is not part of the language definition.

Input

The input contains a nonempty sequence of lines which specify a bill. Each input line is either a priced line or a rake line. A priced line starts with a positive integer, not exceeding 1000, followed immediately by a comma and a minus sign. Optionally, the minus sign is followed by a nonzero number of vertical bars. A rake line contains nonzero number of vertical bars and no other symbols. In the whole bill, the vertical bar is represented as `'|'`, ascii character 124. Each line contains at most 1000 characters, there are no blanks on the line. There are at most 1000 input lines. The input does not contain any empty line. All prices are expressed in the same monetary units.

Output

Output a single number — the bill total rounded up to the nearest 10s. The number should be in the `<PUB_INTEGER>` format, that is, it should be immediately followed by a comma and a minus sign.

Example

Input	Output
|||| 123,-|||	540,-
||| 12,-| ||| 12,-|| 10,-|	300,-
| 8,-|	50,-

Problem C. Beer Coasters

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

Often, in an average pub, the beer is served in a mug which is, quite obviously, wet from inside, but also wet from outside. The barkeeper typically has no capacity to dry freshly washed mugs. To protect the table and the (optional!) tablecloth, beer coasters are used in most pubs. Originally, the beer coasters were round, nowadays you can find a variety of different shapes. Despite being perceived as slightly unorthodox, square and even rectangular coasters are manufactured and used daily in many self-respecting restaurants, taverns, beer houses and drinkeries of high and low profiles.

A research related to the well-known satiric Ig Nobel Prize is being conducted in local pubs. The research aims to measure the rate of wear of the rectangular coasters. The rate of wear depends also on how much a coaster is getting wet from the contact with the wet bottom of beer mugs. That, in turn, depends on the exact area of contact between the mug and the coaster. The exact position of the coaster and the mug on the table is recorded each time the mug is put on the coaster.

Many sophisticated calculations in the research take as input the area of contact, which has to be calculated by a suitable computer program. You are to write such a program.

Input

The input contains 7 space-separated integers on a single line. The first three integers X, Y , R describe the coordinates of the beer mug bottom on the table. The center of the mug bottom is at $(X,\ Y)$ and the radius of the mug bottom is R. The next four integers A_x, A_y, B_x, B_y describe the coordinates of two opposite corners, $(A_x,\ A_y)$ and $(B_x,\ B_y)$, of the coaster on the table. The coaster is a rectangle placed with its sides parallel to the coordinate axes. All coordinates are in the range from -1000 to 1000, the radius is in the range from 1 to 1000. The radius and the coordinates are expressed in the same units of length.

Output

Output a single decimal number with precision of 4 digits after the decimal point, the area of contact between the coaster and the beer mug.

Example

Input	Output
`-1 0 2 -1 -2 3 2`	`6.2832`

Problem D. Beer Flood System

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

Triceratops brewery uses an intricate automated system of pipes to deliver their beer to many local pubs, restaurants and other points of interest. The system is called the Beer flood system. Recently, the brewery is planning its major overhaul.

The system consists of one source station, one collector station, pumping stations and pipes. The source station is located in the brewery. All pumping stations are located in pubs, restaurants, etc. The location of the collector station is somewhat mysterious, however, it is not important for the system function.

The pipes connect some number of stations. The source station is connected to at least one other station and also the collector station is connected to at least one other station. In the extreme case, when the collector station is identical to the source station, there are no pumping stations and no pipes in the system.

The beer flows from the source station through the pipes to other stations where some of it may be (and often is) consumed. Remaining beer which is not consumed in a pumping station automatically continues to flow through the pipes to other pumping station or stations or to the collector station.

The direction of beer flow in the system is always such that the beer which has left a station cannot return to the same station again. A circular flow of beer through any subset of stations never appears in the system. Also, the direction of beer flow between any pair of connected stations is fixed and does not change over time. The size of the beer flow from the source station and through the system is sufficient to supply all pumping stations. There is always some flow of beer through each pipe in the system.

The current Beer flood system was built a long time ago when computer-based optimization was only a dream of the future. Later it became clear that very probably some carefully chosen redundant pipes may be removed from the system and beer flow through the remaining pipes adjusted in such way that the system main features will be preserved. In particular, all pumping stations will still receive a sufficient flow of beer from the source station and all beer unconsumed in pumping stations will still reach the collector. Also, there will be again some flow of beer through each pipe in the system and the direction of the beer flow will not change in any of the remaining pipes. The capacity of all pipes is so big that the remaining pipes need not to be expanded, even though the beer flow through some of them is increased.

Given the topology of the Beer flood system, calculate the maximum number of pipes that can be removed from the system.

Input

The first line contains two integers N, M ($1 \leq N \leq 2000,\ 0 \leq M \leq 5000$). N is the number of all stations in the Beer flood system, including the source station and the collector station. M is the number of pipes in the system. Stations are labeled by integers 1, 2, ..., N. Next, M lines follow, each specifies one pipe and the direction of beer flow through it. The line contains two integers X and Y ($1 \leq X,\ Y \leq N$; $X \neq Y$), beer flows from station X to station Y. All pipes connect unique pairs of stations, no pipe connects a station to itself. In the input, there is exactly one station that receives no flow, it is the source station. Also, from exactly one station no beer flows to any other station, it is the collector station.

Output

Output a single integer specifying the maximal number of pipes that can be removed from the Beer flood system.

Example

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

Problem E. Beer Can Game

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

The Beer Can Rows game is played by a single player whose goal is to modify a given arrangement of given beer cans and wooden tokens in the minimum number of moves to obtain an arrangement satisfying a specific condition.

A beer can row is a straight row of objects each of which is either a beer can or a wooden token. Different tokens may have different values, the value of a token is printed on the token. The beginning and the end of each row are clearly marked.

There are two parallel beer can rows prepared for the player by the game master. Typically, the rows are located in some distance from the location of the game master. This configuration is essential for monitoring the progress of the game. The brands of all beer cans which appear in the prepared beer can rows are available. There might be some additional brands available which do not appear in the prepared beer can rows. All brands are available in an unlimited number of cans. All cans of all brands are under the supervision of the game master.

The player begins the game at the game master's location. From there the player proceeds to the prepared beer can rows where he or she starts to perform moves of the game, one after another.

There are three possible moves a player can perform, Insert can, Expand token, Remove can.

1. Insert can. The player goes to the game master, asks for a beer can of any available brand and receives it, carries the can back to the beer can rows and inserts it to any row at any place or puts it at the beginning or at the end of any row.

2. Expand token. The player takes a token from any beer can row and brings it to the game master who exchanges the token for some beer cans. The number of cans is equal to the number printed on the token. The player can choose any mix of brands of the cans. Next, the player carries the cans back to the beer can row from which the token was removed and puts the cans, in any order, into the row at the place of the removed token.

3. Remove can. The player takes out any can from any of the beer can rows, carries it to the game master and throws it into the dedicated litter bin in his vicinity. Then the player immediately returns to the beer can rows.

The player is obliged to keep each beer can row arranged neatly in a straight line for the entire duration of the game to avoid any uncertainty regarding the order of objects in the row.

The goal of the game is to obtain two identical beer can rows. Two rows are considered to be identical if they contain only beer cans, the number of the cans in both rows is the same and the brand of the k-th can in one row is the same as the brand of the k-th can in the other row, for all values of k in the range from 1 to the length of the rows.

Find the minimum number of moves player needs to finish the Beer Can Rows game.

Input

The input consists of two lines which represent two initial beer can rows. Each line contains only lowercase characters (`a - z`) and decimal digits `0 - 9`. Each character represents one beer can brand, different characters represent different brands. Each brand available in the game is represented by one lowercase character, the character may or may not appear in the input lines. Each digit represents one token, the

value of the digit is equal to the value printed on the token. Any input line may contain either only lowercase characters or only decimal digits.

The first input line is at least 1 and at most 10^4 characters long. The second input line is at least 1 and at most 10^3 characters long. There are no more than 100 digits in either of the input lines.

Output

Output a single integer specifying the minimal number of moves a player must perform to obtain two identical rows of cans.

Example

Input	Output
beer 4	1
beer5ing 4drinking	2

Problem F. Beer Marathon

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

In the booths version of the popular annual Beer Marathon, many beer booths (also called beer stalls or beer stands) are installed along the track. A prescribed number of visits to different beer booths is a part of the race for all contestants. The distances between any two consecutive beer booths should be exactly the same and equal to the particular value specified in the race rules.

The company responsible for beer booths installation did the sloppy work (despite being excessively paid), left the beer booths on more or less random positions along the track and departed from the town. Fortunately, thanks to advanced AI technology, the company was able to report the exact positions of the beer booths along the track, measured in meters, with respect to the particular anchor point on the track. The marathon committee is going to send out the volunteers to move the beer booths to new positions, consistent with the race rules.

They want to minimize the total number of meters by which all beer booths will be moved. The start line and the finishing line of the race will be chosen later, according to the resulting positions of the beer booths.

Input

The first line of input contains two integers N and K ($1 \leq N, K \leq 10^6$), where N is the number of beer booths and K is the prescribed distance between the consecutive beer booths. The second line of input contains N pairwise different integers $x_1, \ldots, x_N$ ($-10^6 \leq x_i \leq 10^6$), the original positions, in meters, of the beer booths relative to the anchor point. The positive values refer to the positions past the anchor point and the negative values refer to the positions before the anchor point.

Output

Output a single integer — the minimal total number of meters by which the beer booths should be moved to satisfy the race rules.

Example

Input	Output
3 1 2 5 7	3
10 4 140 26 69 55 39 64 2 89 78 421	511

Problem G. Beer Mugs

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

Damian is a beer mug collector. His collection fills most of the shelves in his vintage wooden cabinet where all mugs are proudly displayed. The mugs are of various brands. There might be, and often are, more mugs of the same brand in the collection.

Mugs on a shelf in Damian's collection always form a single symmetric row. Specifically, the symmetry of the row means that the sequence of particular mug brands in the row is the same when the mugs are being admired one by one from left to right and when the mugs are being admired one by one from right to left. There is still one empty shelf in the cabinet and Damian looks for an opportunity to fill it with a new set of mugs.

The widely recognized Mastodon brewery (admired for its Woolly Mammoth beer) organizes annually the so-called beer season. Participants of the season are engaged in daily beer brewing activities and are rewarded each day by a special collector mug. Each day in the season is assigned a particular mug brand. The mug brands for all days in the season are known in advance, some brands may appear repeatedly in the season.

A participant may subscribe for the whole season or just for a part of the season. However, all days of his or her participation have to be in one uninterrupted sequence of days, a participant cannot leave the season and then come back again after some days of absence.

Damian is keen to take part in the beer season. He decided that the set of mugs he brings home should be suited for his display without adding or removing any mug, and that the set should be as big as possible.

Given the list of mug brands provided by the brewery for all days in the beer season, find the size of the biggest set of mugs suitable for Damian's display which can be obtained by subscribing to some appropriately chosen part of the beer season.

Input

The first input line contains integer N ($1 < N \leq 3 \cdot 10^5$) the number of days in the brewery beer season. The next line contains N characters representing the list of all beer mug brands offered in the season, day by day. The list is naturally ordered from the first day to the last day of the season. Each brand is coded by a single lower case letter, from '`a`' to '`t`'. The list contains no blanks.

Output

Print a single integer representing the size of the biggest set of mugs which Damian can bring home from the brewery beer season and which is suitable, without changes, for his collection display.

Example

Input	Output
6 abcabc	6
20 ghjahjghsajdjhlfslja	7
12 aabbccddabcd	9

Problem H. Screamers in the Storm

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

As you might remember from your first years in school, the human race invented beer brewing at least about 7000 years ago. The total amount of beer consumed from those times must be monumental and surely the rate of consumption is not going to shrink in the coming millennia.

To celebrate these facts, we invite you to implement a game, seemingly unrelated to beer brewing. It is quite possible, however, that after a successful implementation you might feel a little dizzy, just as if you have had a little bit more than your daily dose of beer...

The game is played on a rectangular $M \times N$ grid consisting of square tiles of different types.

Animals and food

There is some number of animals on the grid, each animal occupies exactly one tile.

Grass grows, sheep eat grass, wolves eat sheep, and both wolves and sheep can die of starvation.

Turns and animal actions

Each turn consists of actions in the following order:

1. All animals on the grid move. Each wolf moves to a neighbouring tile in the east (to the right). If the wolf's move is not possible, the wolf moves to the westernmost tile in its row. Each sheep moves to a neighbouring tile in the south (down). If the sheep's move is not possible, the sheep moves to the northernmost tile in its column.
2. If a wolf and a sheep occupy the same tile, the wolf eats the sheep and the tile is changed to a *Soil with carcass* tile.
3. If a sheep is located on a *Soil with grass* tile, the sheep eats the grass and the tile is changed to a *Soil* tile.
4. If a wolf hasn't eaten in any of the last 10 turns including the current turn, it dies and the tile is changed to a *Soil with carcass* tile.
5. If a sheep didn't eat in any of the last 5 turns including the current turn, it dies and the tile is changed to a *Soil with carcass* tile.

Types of tiles and their changes

There are three types of tiles. The type of a tile may be changed in the course of the game.

1. *Soil* tile: After 3 turns after the beginning of the game or after 3 turns after the tile became a *Soil* tile, the tile becomes a *Soil with grass* tile.
2. *Soil with grass* tile: If the grass is eaten by a sheep, the tile immediately becomes a *Soil* tile. The grass will grow again at the tile after 3 turns.
3. *Soil with carcass* tile: Whenever an animal dies on a tile of any type, the tile immediately becomes a *Soil with carcass* tile. Animals can still move to this tile, but the grass will never grow on this tile again. As the game progresses, more carcasses may accumulate on the tile.

Input

First line of the input contains three integers T, N and M ($1 \leq T \leq 100,\ 1 \leq M, N \leq 20$), where T is the number of turns, M is the number of rows and N is the number of columns of the grid. The following M lines contain N characters each. Characters denote the types of tiles:

- `.` (dot character) denotes a *Soil* tile
- `S` denotes a *Soil* tile with a sheep on it
- `W` denotes a *Soil* tile with a wolf on it

Output

Output M lines each containing N characters describing the state of the grid after the end of the T-th turn. If there is an animal on a tile, output:

- `W` for a tile with a wolf on it
- `S` for a tile with a sheep on it

Otherwise, output:

- `*` for a *Soil with carcass* tile
- `#` for a *Soil with grass* tile
- `.` (dot character) for a *Soil* tile

Example

Input	Output
6 6 5 ..S.. ..... .S... ....W .S...	##S## ##### ##### #.### W*.## #S.##
14 3 3 S.. W.. ...	.## #*# S##
2 3 1 S . .	. . S
3 3 1 S . .	S # #
4 3 1 S . .	# S #
5 3 1 S . .	# . S

Problem I. Sixpack

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

The glossy fashionable National Gentlemen and Ladies Beer Magazine runs monthly contests targeted primarily on young IT experts who are, of course, also the predominant subscribers of the Magazine. A contest is based on the so-called Sixpack puzzle, printed on the first page of the Magazine on a stylish beer foam background. The puzzle consists of a rectangular grid with two rows and three or more columns. Some cells in the grid are empty, some contain a single decimal digit, different cells might contain different digits. In extreme cases, the grid might be empty or it might be completely filled. Any three consecutive columns in the grid are called a sixpack, hence the name of the puzzle.

The grid is accompanied with another integer K which is an integral part of the puzzle.

The task of the reader is to fill each empty cell in the grid with a single digit, in such a way that the sum of values in each sixpack is equal to K. Different cells may contain different digits. Next, the reader sends his or her solution of the puzzle to the advisory board of the magazine. The board keeps track of all the solutions they receive. When the reader's solution is the same as some other solution received earlier by the board, the reader does not win any prize. When the reader's solution differs from all solutions received by the board so far, the reader wins a package of real beer sixpacks of a quality beer brand. The number of sixpacks in the package is equal to the number of different puzzle solutions which were at the possession of the board immediately after the moment the board received the reader's solution.

Two solutions are considered different if they differ in the contents of at least one cell in the grid at the same position.

A reader can send in at most one solution. Any additional solution received from the same reader is always dismissed. The secretary of the board guarantees that the board never receives two or more solutions at the same moment in time.

Given a particular Sixpack puzzle, calculate the maximum number of beer sixpacks a Magazine reader can win in the respective contest. The Magazine is so popular you can be sure that the number of magazine readers is higher than the number of unique solutions of the puzzle.

Input

The input specifies one Sixpack puzzle. The first input line contains three integers N ($3 \leq N \leq 10^5$), K ($0 \leq K \leq 100$) and M ($0 \leq M \leq 2 \cdot 10^5$). N is the number of columns in the grid, K specifies the prescribed sum in each sixpack and M is the number of predefined values in the grid. Each of the following M lines contains three integers C ($0 \leq C \leq N-1$), R ($0 \leq R \leq 1$) and V ($0 \leq V \leq 9$). C and R specify the column and the row of the cell in the grid, V is the predefined value in that cell. Each cell's value in the grid is specified at most once.

Output

Output the maximum possible number of sixpacks a magazine reader can win. Print this number modulo $10^9 + 7$.

Example

Input	Output
3 2 0	21
5 17 2 0 1 5 4 0 2	11580

Problem J. Beer Vision

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

We are given a (drunken) image of stars as seen by a drunken man lying on his back on the grass in the vicinity of a closed pub late in the evening. His image is a blend of one original (sober) image, and a copy of the same image shifted by some fixed $(X,\ Y)$ nonzero vector. Only the resulting blended image is perceived by the drunkard. Neither the original sober image nor the shift vector are available to him and to us, unfortunately.

An act of humanity would be to restore his perceived image to the version seen by his sober fellow citizens.

Given an image, write a program which calculates how many distinct $(X,\ Y)$ vectors exist such that the drunken image can be created by merging some original sober image with its copy shifted by the vector.

Note that if the images of two different stars — one in the original image and the other in its shifted copy — overlap in the blended image, then the drunken image, which is also the input of the program, contains only one entry for this position.

Input

The first line of input contains an integer N $(0 < N \leq 1000)$, the number of stars in the blended (drunken) image. Next, there are N lines, each with two space-separated integers X_i, Y_i $(-1000 \leq X_i, Y_i \leq 1000)$ describing the position of a star. All stars are regarded to be points with no dimensions.

Output

Print the number of distinct vectors with non-zero length which can be applied to an unknown sober picture to produce the input drunken image. The unknown image might be different in different cases.

Example

Input	Output
5 0 0 1 1 2 2 2 0 3 1	2
3 0 0 0 1 1 0	0