Problem A. Billiard

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

You have a billiard table. The playing area of the table is rectangular. This billiard table is special as it has no pockets, and the playing area is completely surrounded with a cushion.

You succeeded in producing a ultra-precision billiards playing robot. When you put some balls on the table, the machine hits one of those balls. The hit ball stops after 10,000 unit distance in total moved.

When a ball collided with the cushion of the table, the ball takes the orbit like mirror reflection. When a ball collided with the corner, the ball bounces back on the course that came.

Your mission is to predict which ball collides first with the ball which the robot hits .

Input

First line of a input file contains an positive integer n, which represents the number of balls on the table ($2 \leq n \leq 11$). The next line contains five integers (w, h, r, v_x, v_y) separated by a single space, where w and h are the width and the length of the playing area of the table respectively ($4 \leq w, h \leq 1{,}000$), r is the radius of the balls ($1 \leq r \leq 100$). The robot hits the ball in the vector (v_x, v_y) direction ($-10{,}000 \leq v_x, v_y \leq 10{,}000$ and $(v_x, v_y) \neq (0, 0)$).

The following n lines give position of balls. Each line consists two integers separated by a single space, (x_i, y_i) means the center position of the i-th ball on the table in the initial state ($r < x_i < w - r$, $r < y_i < h - r$). $(0, 0)$ indicates the position of the north-west corner of the playing area, and (w, h) indicates the position of the south-east corner of the playing area. You can assume that, in the initial state, the balls do not touch each other nor the cushion.

The robot always hits the first ball in the list. You can assume that the given values do not have errors.

Output

Print the index of the ball which first collides with the ball the robot hits. When the hit ball collides with no ball until it stops moving, print "`-1`".

You can assume that no more than one ball collides with the hit ball first, at the same time.

It is also guaranteed that, when r changes by eps ($eps < 10^{-9}$), the ball which collides first and the way of the collision do not change.

Examples

standard input	standard output
3 26 16 1 8 4 10 6 9 2 9 10	3
3 71 363 4 8 0 52 238 25 33 59 288	-1

Problem B. Counting 1's

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	512 mebibytes

Let $b_i(x)$ be the i-th least significant bit of x, i.e. the i-th least significant digit of x in base 2 ($i \geq 1$). For example, since $6 = (110)_2$, $b_1(6) = 0$, $b_2(6) = 1$, $b_3(6) = 1$, $b_4(6) = 0$, $b_5(6) = 0$, and so on.

Let A and B be integers that satisfy $1 \leq A \leq B \leq 10^{18}$, and k_i be the number of integers x such that $A \leq x \leq B$ and $b_i(x) = 1$.

Your task is to write a program that determines A and B for a given $\{k_i\}$.

Input

The input consists of up to 35 000 test cases. The first line of each case contains an integer n ($1 \leq n \leq 64$). Then n lines follow, each of which contains k_i ($0 \leq k_i \leq 2^{63} - 1$). For all $i > n$, $k_i = 0$.

Output

Print one line for each case.

- If A and B can be uniquely determined, output A and B. Separate the numbers by a single space.
- If there exists more than one possible pair of A and B, output "`Many`" (without quotes).
- Otherwise, i.e. if there exists no possible pair, output "`None`" (without quotes).

Examples

standard input	standard output
3 2 2 1	1 4
4 1 1 1 1	Many
3 4 2 2	None

Problem C. Dictionary

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

We found a dictionary of the Ancient Civilization Mayo (ACM) during excavation of the ruins. After analysis of the dictionary, we revealed they used a language that had not more than 26 letters. So one of us mapped each letter to a different English alphabet and typed all the words in the dictionary into a computer.

How the words are ordered in the dictionary, especially whether they are ordered lexicographically, is an interesting topic to many people. As a good programmer, you are requested to write a program to judge whether we can consider the words to be sorted in a lexicographical order.

Note: In a lexicographical order, a word always precedes other words it is a prefix of. For example, "`ab`" precedes "`abc`", "`abde`", and so on.

Input

Input file consists of $n + 1$ lines. The first line contains an integer that indicates n ($1 \leq n \leq 500$). The i-th line of the following n lines contains non-empty $string_i$, which consists of up to 10 English lowercase letters.

Output

If all words in the input file can be considered to be ordered lexicographically, print "`yes`". Otherwise, print "`no`".

Example

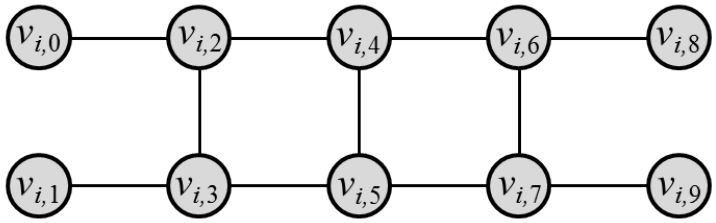
standard input	standard output
4 cba cab b a	yes
3 bca ab a	no
5 abc acb b c c	yes
5 abc acb c b b	no

Problem D. Hashigo Sama

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

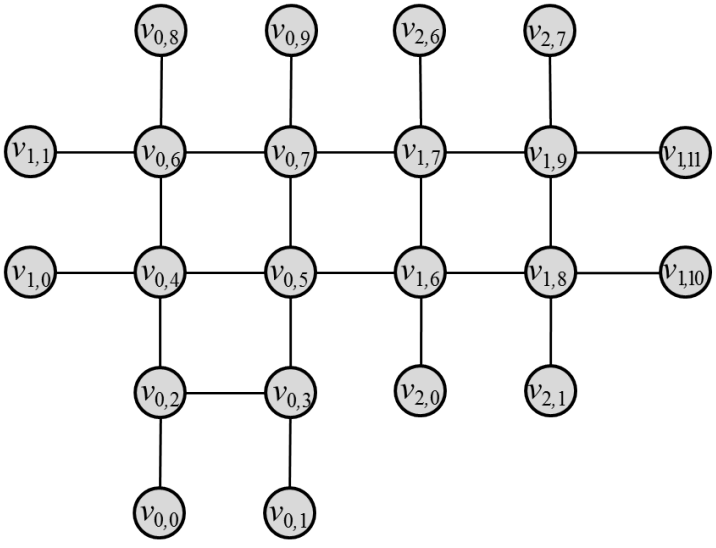
Chelsea is a modern artist. She decided to make her next work with ladders. She wants to combine some ladders and paint some beautiful pattern.

A ladder can be considered as a graph called *hashigo*. There are n *hashigos* numbered from 0 to $n-1$. *Hashigo* i of length l_i has $2l_i+6$ vertices $v_{i,0}, v_{i,1}, ..., v_{i,2l_i+5}$ and has edges between the pair of vertices $(v_{i,j}, v_{i,j+2})$ $(0 \le j \le 2l_i+3)$ and $(v_{i,2j}, v_{i,2j+1})$ $(1 \le j \le l_i+1)$. The figure below is example of a *hashigo* of length 2. This corresponds to the graph given in the first dataset in the sample input.



Two *hashigos* i and j are combined at position p $(0 \le p \le l_i-1)$ and q $(0 \le q \le l_j-1)$ merging each pair of vertices $(v_{i,2p+2}, v_{j,2q+2})$, $(v_{i,2p+3}, v_{j,2q+4})$, $(v_{i,2p+4}, v_{j,2q+3})$ and $(v_{i,2p+5}, v_{j,2q+5})$.

Chelsea performs this operation $n-1$ times to combine the n *hashigos*. After this operation, the graph should be connected and the maximum degree of the graph should not exceed 4. The figure below (flattened for simpilcity) is a example of the graph obtained by combining three *hashigos*. This corresponds to the graph given in the second dataset in the sample input.



(note that while merging two *hashigos* no extra connections with other *hashigos* are created (illusion of those extra connections may appear while looking at this figure)

Now she decided to paint each vertex by black or white with satisfying the following condition:

- The number of vertices in each of the components formed by the connected vertices painted by the same color is less than or equals to k.

She would like to try all the patterns and choose the best. However, the number of painting way can be very huge. Since she is not good at math nor computing, she cannot calculate the number. So please help her with your superb programming skill!

Input

The first line of input file contains two integers n ($1 \leq n \leq 30$) and k ($1 \leq k \leq 8$).

The next line contains n integers l_i ($1 \leq l_i \leq 30$), each denotes the length of *hashigo* i.

The following $n-1$ lines each contains four integers f_i ($0 \leq f_i \leq n-1$), p_i ($0 \leq p_i \leq l_{f_i} - 1$), t_i ($0 \leq t_i \leq n-1$), q_i ($0 \leq q_i \leq l_{t_i} - 1$). It represents the *hashigo* f_i and the *hashigo* t_i are combined at the position p_i and the position q_i. You may assume that the graph obtained by combining n *hashigos* is connected and the degree of each vertex of the graph does not exceed 4.

Output

Print the number of different colorings modulo 1,000,000,007 in a line.

Examples

standard input	standard output
1 5 2	708
3 7 2 3 1 0 1 1 0 1 2 2 0	1900484
2 8 5 6 0 2 1 2	438404500
2 8 1 1 0 0 1 0	3878
2 2 2 2 0 1 1 0	496
2 3 3 3 0 2 1 1	14246
2 4 3 1 1 0 0 1	9768

Problem E. Texas Hold 'em

Input file:	`standard input`
Output file:	`standard outptu`
Time limit:	2 seconds
Memory limit:	512 mebibytes

Texas Hold 'em is one of the standard poker games, originated in Texas, United States. It is played with a standard deck of 52 cards, which has 4 *suits* (spades, hearts, diamonds and clubs) and 13 *ranks* (A, K, Q, J and 10–2), without jokers.

With betting aside, a game goes as described below.

At the beginning each player is dealt with two cards face down. These cards are called *hole cards* or *pocket cards*, and do not have to be revealed until the showdown. Then the dealer deals three cards face up as *community cards*, i.e. cards shared by all players. These three cards are called the *flop*. The flop is followed by another community card called the *turn* then one more community card called the *river*.

After the river, the game goes on to the *showdown*. All players reveal their hole cards at this point. Then each player chooses five out of the seven cards, i.e. their two hole cards and the five community cards, to form a *hand*. The player forming the strongest hand wins the game. There are ten possible kinds of hands, listed from the strongest to the weakest:

- *Royal straight flush*: A, K, Q, J and 10 of the same suit. This is a special case of straight flush.
- *Straight flush*: Five cards in sequence (e.g. 7, 6, 5, 4 and 3) and of the same suit.
- *Four of a kind*: Four cards of the same rank.
- *Full house*: Three cards of the same rank, plus a pair of another rank.
- *Flush*: Five cards of the same suit, but not in sequence.
- *Straight*: Five cards in sequence, but not of the same suit.
- *Three of a kind*: Just three cards of the same rank.
- *Two pairs*: Two cards of the same rank, and two other cards of another same rank.
- *One pair*: Just a pair of cards (two cards) of the same rank.
- *High card*: Any other hand.

For the purpose of a sequence, J, Q and K are treated as 11, 12 and 13 respectively. A can be seen as a rank either next above K or next below 2, thus both A-K-Q-J-10 and 5-4-3-2-A are possible (but not 3-2-A-K-Q or the likes).

If more than one player has the same kind of hand, ties are broken by comparing the ranks of the cards. The basic idea is to compare first those forming sets (pairs, triples or quads) then the rest cards one by one from the highest-ranked to the lowest-ranked, until ties are broken. More specifically:

- *Royal straight flush*: (ties are not broken)
- *Straight flush*: Compare the highest-ranked card.
- *Four of a kind*: Compare the four cards, then the remaining one.
- *Full house*: Compare the three cards, then the pair.
- *Flush*: Compare all cards one by one.
- *Straight*: Compare the highest-ranked card.
- *Three of a kind*: Compare the three cards, then the remaining two.
- *Two pairs*: Compare the higher-ranked pair, then the lower-ranked, then the last one.
- *One pair*: Compare the pair, then the remaining three.
- *High card*: Compare all cards one by one.

The order of the ranks is A, K, Q, J, 10, 9, ..., 2, from the highest to the lowest, except for A next to 2 in a straight regarded as lower than 2. Note that there are exceptional cases where ties remain. Also note that the suits are not considered at all in tie-breaking.

Here are a few examples of comparison (note these are only intended for explanatory purpose; some combinations cannot happen in Texas Hold 'em):

- J-J-J-6-3 and K-K-Q-Q-8.

The former beats the latter since three of a kind is stronger than two pairs.

- J-J-J-6-3 and K-Q-8-8-8.

Since both are three of a kind, the triples are considered first, J and 8 in this case. J is higher, hence the former is a stronger hand. The remaining cards, 6-3 and K-Q, are not considered as the tie is already broken.

- Q-J-8-6-3 and Q-J-8-5-3.

Both are high cards, assuming hands not of a single suit (i.e. flush). The three highest-ranked cards Q-J-8 are the same, so the fourth highest are compared. The former is stronger since 6 is higher than 5.

- 9-9-Q-7-2 and 9-9-J-8-5.

Both are one pair, with the pair of the same rank (9). So the remaining cards, Q-7-2 and J-8-5, are compared from the highest to the lowest, and the former wins as Q is higher than J.

Now suppose you are playing a game of Texas Hold 'em with one opponent, and the hole cards and the flop have already been dealt. You are surprisingly telepathic and able to know the cards the opponent has. Your ability is not, however, as strong as you can predict which the turn and the river will be.

Your task is to write a program that calculates the probability of your winning the game, assuming the turn and the river are chosen uniformly randomly from the remaining cards. You and the opponent always have to choose the hand strongest possible. Ties should be included in the calculation, i.e. should be counted as losses.

Input

Input file consists of three lines. The first and second lines contain the hole cards of yours and the opponent's respectively. The third line contains the flop, i.e. the first three community cards. These cards are separated by spaces.

Each card is represented by two characters. The first one indicates the suit: `S` (spades), `H` (hearts), `D` (diamonds) or `C` (clubs). The second one indicates the rank: `A`, `K`, `Q`, `J`, `T` (10) or `9`–`2`.

Output

Print the probability in a line. The number may contain an arbitrary number of digits after the decimal point, but should not contain an absolute error greater than 10^{-6}.

Examples

standard input	standard outptu
SA SK DA CA SQ SJ ST	1.00000000000000000000
SA HA D2 C3 H4 S5 DA	0.34444444444444444198
HA D9 H6 C9 H3 H4 H5	0.63030030303030325391

Problem F. Magical Bridges

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

A magician lives in a country which consists of N islands and M bridges. Some of the bridges are magical bridges, which are created by the magician. Her magic can change all the lengths of the magical bridges to the same non-negative integer simultaneously.

This country has a famous 2-player race game. Player 1 starts from island S_1 and Player 2 starts from island S_2. A player who reached the island T first is a winner.

Since the magician loves to watch this game, she decided to make the game most exiting by changing the length of the magical bridges so that the difference between the shortest-path distance from S_1 to T and the shortest-path distance from S_2 to T is as small as possible. We ignore the movement inside the islands.

Your job is to calculate how small the gap can be.

Note that she assigns a length of the magical bridges before the race starts *once*, and does not change the length *during the race*.

Input

Input file obeys *the* following format. Every number in the inputs is integer.

$N\ M\ S_1\ S_2\ T$
$a_1\ b_1\ w_1$
$a_2\ b_2\ w_2$
...
$a_M\ b_M\ w_M$

(a_i, b_i) indicates the bridge i connects the two islands a_i and b_i.

w_i is either a non-negative integer or a letter '`x`' (quotes for clarity). If w_i is an integer, it indicates the bridge i is normal and its length is w_i. Otherwise, it indicates the bridge i is magical.

You can assume the following:

- $1 \leq N \leq 1{,}000$
- $1 \leq M \leq 2{,}000$
- $1 \leq S_1, S_2, T \leq N$
- S_1, S_2, T are all different.
- $1 \leq a_i, b_i \leq N$
- $a_i \neq b_i$
- For all normal bridge i, $0 \leq w_i \leq 1{,}000{,}000{,}000$
- The number of magical bridges ≤ 100

Output

Print one number — the minimal gap.

Examples

standard input	standard output
3 2 2 3 1 1 2 1 1 3 2	1
4 3 1 4 2 2 1 3 2 3 x 4 3 x	1

Problem G. Stack Maze

Input file:	standard input
Output file:	standard output
Time limit:	2 seconds
Memory limit:	512 mebibytes

There is a maze which can be described as a $W \times H$ grid. The upper-left cell is denoted as (1, 1), and the lower-right cell is (W, H). You are now at the cell (1, 1) and have to go to the cell (W, H). However, you can only move to the right adjacent cell or to the lower adjacent cell. The following figure is an example of a maze.

```
...#......
a###.#####
.bc...A...
##.#C#d#.#
.#B#.#.###
.#...#e.D.
.#A..###.#
..e.c#..E.
####d###.#
#....#.#.#
##E...d.C.
```

In the maze, some cells are free (denoted by '`.`') and some cells are occupied by rocks (denoted by '`#`'), where you cannot enter. Also there are jewels (denoted by lowercase alphabets) in some of the free cells and holes to place jewels (denoted by uppercase alphabets). Different alphabets correspond to different types of jewels, i.e. a cell denoted by '`a`' contains a jewel of type A, and a cell denoted by '`A`' contains a hole to place a jewel of type A. It is said that, when we place a jewel to a corresponding hole, something happy will happen.

At the cells with jewels, you can choose whether you pick a jewel or not. Similarly, at the cells with holes, you can choose whether you place a jewel you have or not. You can't put more than one jewel into a single hole. Initially you do not have any jewels. You have a very big bag, so you can bring arbitrarily many jewels. However, your bag is a stack, that is, you can only place the jewel that you picked up last.

On the way from cell (1, 1) to cell (W, H), how many jewels can you place to correct holes?

Input

First line of input file contains two integers H and W — the height and width of the grid, respectively. You may assume that $1 \leq W, H \leq 50$. The rest of the input file consists of H lines, each of which is composed of W letters. Each letter C_{ij} specifies the type of the cell (i, j) as described before. It is guaranteed that C_{11} and C_{WH} are never '`#`'.

You may also assume that each lowercase or uppercase alphabet appear at most 10 times.

Output

Output the maximum number of jewels that you can place to corresponding holes. If you cannot reach the cell (W, H), output -1.

Examples

standard input
3 3 ac# b#C .BA
standard output
2

standard input
3 3 aaZ a#Z aZZ
standard output
0

standard input
3 3 ..# .#. #..
standard output
-1

standard input
1 50 abcdefghijklmnopqrstuvwxyYXWVUTSRQPONMLKJIHGFEDCBA
standard output
25

standard input
1 50 aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyY
standard output
25

standard input
1 50 abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXY
standard output
1

standard input
1 50 aaaaaaaaaabbbbbbbbbbccccccCCCCCBBBBBBBBBBAAAAAAAAAA
standard output
25

standard input
10 10 ...#...... a###.##### .bc...A... ##.#C#d#.# .#B#.#.### .#...#e.D. .#A..###.# ..e.c#..E. ####d###.# ##E...D.C.
standard output
4

Problem H. Median Tree

Input file:	standard input
Output file:	standard output
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a connected undirected graph which has even numbers of nodes. A connected graph is a graph in which all nodes are connected directly or indirectly by edges.

Your task is to find a spanning tree whose median value of edges' costs is minimum. A spanning tree of a graph means that a tree which contains all nodes of the graph.

Input

The first line contains an even number n ($2 \leq n \leq 1{,}000$) and an integer m ($n - 1 \leq m \leq 10{,}000$). n is the nubmer of nodes and m is the number of edges in the graph.

Then m lines follow, each of which contains s_i ($1 \leq s_i \leq n$), t_i ($1 \leq s_i \leq n, t_i \neq s_i$) and c_i ($1 \leq c_i \leq 1{,}000$). This means there is an edge between the nodes s_i and t_i and its cost is c_i. There is no more than one edge which connects s_i and t_i.

Output

Print one integer — the median value.

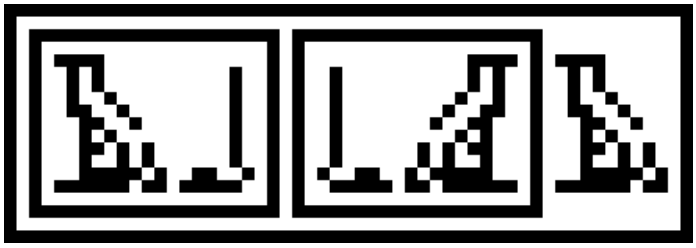
Examples

standard input	standard output
2 1 1 2 5	5
4 6 1 2 1 1 3 2 1 4 3 2 3 4 2 4 5 3 4 6	2
8 17 1 4 767 3 1 609 8 3 426 6 5 972 8 1 607 6 4 51 5 1 683 3 6 451 3 4 630 8 7 912 3 7 43 4 7 421 3 5 582 8 4 538 5 7 832 1 6 345 8 2 608	421

Problem I. Ancient Commemorative Monolith

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

One day in a forest, Alice found an old monolith.



She investigated the monolith, and found this was a sentence written in an old language. A sentence consists of glyphs and rectangles which surrounds them. For the above example, there are following glyphs.

Notice that some glyphs are flipped horizontally in a sentence.

She decided to transliterate it using ASCII letters assigning an alphabet to each glyph, and '`[`' and '`]`' to rectangles. If a sentence contains a flipped glyph, she transliterates it from right to left. For example, she could assign '`a`' and '`b`' to the above glyphs respectively. Then the above sentence would be transliterated into "`[[ab][ab]a]`".

After examining the sentence, Alice figured out that the sentence was organized by the following structure:

- A sentence ⟨seq⟩ is a sequence consists of zero or more ⟨term⟩s.
- A term ⟨term⟩ is either a glyph or a ⟨box⟩. A glyph may be flipped.
- ⟨box⟩ is a rectangle surrounding a ⟨seq⟩. The height of a box is larger than any glyphs inside.

Now, the sentence written on the monolith is a nonempty ⟨seq⟩. Each term in the sequence fits in a rectangular bounding box, although the boxes are not explicitly indicated for glyphs. Those bounding boxes for adjacent terms have no overlap to each other. Alice formalized the transliteration rules as follows.

Let f be the transliterate function.

Each sequence $s = t_1 t_2 \cdots t_m$ is written either from left to right, or from right to left. However, please note here t_1 corresponds to the leftmost term in the sentence, t_2 to the 2nd term from the left, and so on.

Let's define $\bar{g}$ to be the flipped glyph g. A sequence must have been written from right to left, when a sequence contains one or more single-glyph term which is unreadable without flipping, i.e. there exists an integer i where t_i is a single glyph g, and g is not in the glyph dictionary given separately whereas

$\bar{g}$ is. In such cases $f(s)$ is defined to be $f(t_m)f(t_{m-1})\cdots f(t_1)$, otherwise $f(s) = f(t_1)f(t_2)\cdots f(t_m)$. It is guaranteed that all the glyphs in the sequence are flipped if there is one or more glyph which is unreadable without flipping.

If the term t_i is a box enclosing a sequence s', $f(t_i) =$ '[' $f(s')$ ']'. If the term t_i is a glyph g, $f(t_i)$ is mapped to an alphabet letter corresponding to the glyph g, or $\bar{g}$ if the sequence containing g is written from right to left.

Please make a program to transliterate the sentences on the monoliths to help Alice.

Input

Each dataset has the following format.

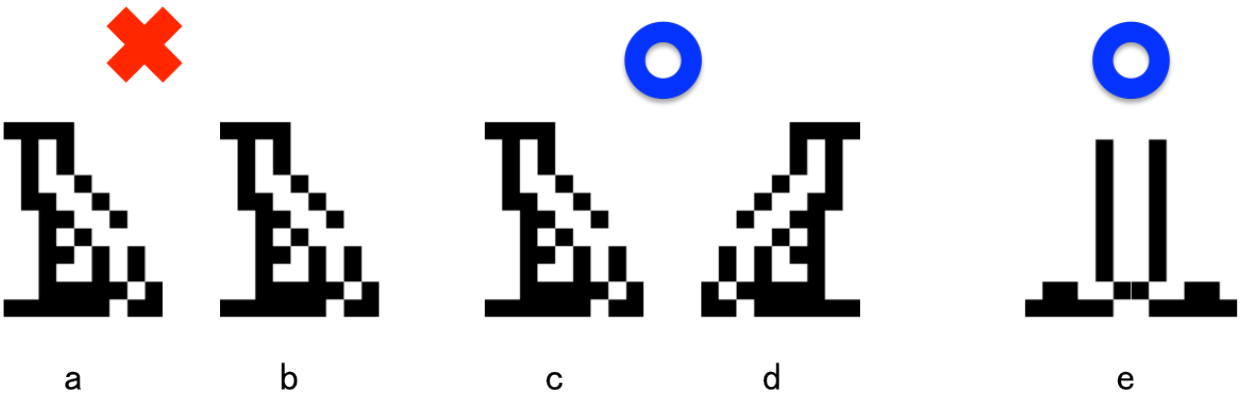
n m
$glyph_1$
...
$glyph_n$
$string_1$
...
$string_m$

n ($1 \leq n \leq 26$) is the number of glyphs and m ($1 \leq m \leq 10$) is the number of monoliths. $glyph_i$ is given by the following format.

c h w
$b_{11}...b_{1w}$
...
$b_{h1}...b_{hw}$

c is a lower-case alphabet that Alice assigned to the glyph. h and w ($1 \leq h \leq 15$, $1 \leq w \leq 15$) specify the height and width of the bitmap of the glyph, respectively. The matrix b indicates the bitmap. A white cell is represented by '.' and a black cell is represented by '*'.

You can assume that all the glyphs are assigned distinct characters. You can assume that every column of the bitmap contains at least one black cell, and the first and last row of the bitmap contains at least one black cell. Every bitmap is different to each other, but it is possible that a flipped bitmap is same to another bitmap. Moreover there may be symmetric bitmaps.



$string_i$ is given by the following format.

h w
$b_{11}...b_{1w}$
...
$b_{h1}...b_{hw}$

h and w ($1 \leq h \leq 100$, $1 \leq w \leq 1000$) is the height and width of the bitmap of the sequence. Similarly to the glyph dictionary, b indicates the bitmap where A white cell is represented by '`.`' and a black cell by '`*`'.

There is no noise: every black cell in the bitmap belongs to either one glyph or one rectangle box. The height of a rectangle is at least 3 and more than the maximum height of the tallest glyph. The width of a rectangle is at least 3.

A box must have a margin of 1 pixel around the edge of it.

You can assume the glyphs are never arranged vertically. Moreover, you can assume that if two rectangles, or a rectangle and a glyph, are in the same column, then one of them contains the other. For all bounding boxes of glyphs and black cells of rectangles, there is at least one white cell between every two of them. You can assume at least one cell in the bitmap is black.

Output

For each monolith, output the transliterated sentence in one line.

Examples

standard input
3 1 a 2 2 .* ** b 2 1 * * c 3 3 *** *.* *** 11 16 **************** *..............* *....*********.* *....*.......*.* *.*..*.***...*.* *.**.*.*.*.*.*.* *....*.***.*.*.* *....*.......*.* *....*********.* *..............* ****************
standard output
[[cb]a]

standard input

```
2 1
a 11 9
****.....
.*.*.....
.*.*.....
.*..*....
.**..*...
..**..*..
..*.*....
..**.*.*.
..*..*.*.
..*****.*
******.**
b 10 6
....*.
....*.
....*.
....*.
....*.
....*.
....*.
....*.
....*.
.**..*
*****.
19 55
*********************************************************
*.....................................................*
*.*******************.*******************...........*
*.*.................*.*.................*...........*
*.*.****............*.*............****.*.****......*
*.*..*.*.........*..*.*..*.........*.*..*..*.*......*
*.*..*.*.........*..*.*..*.........*.*..*..*.*......*
*.*..*..*........*..*.*..*.........*..*..*..*..*.....*
*.*..**..*.......*..*.*..*.......*..**..*..**..*....*
*.*...**..*......*..*.*..*......*..**...*...**..*...*
*.*...*.*........*..*.*..*........*.*...*...*.*.....*
*.*...**.*.*.....*..*.*..*.....*.*.**...*...**.*.*..*
*.*...*..*.*.....*..*.*..*.....*.*..*...*...*..*.*..*
*.*...*****.*..**..*.*.*.*..**..*.*****...*...*****.*.*
*.*.******.**.*****..*.*..*****.**.******.*.******.**.*
*.*.................*.*.................*...........*
*.*******************.*******************...........*
*.....................................................*
*********************************************************
```

standard output

```
[[ab][ab]a]
```

standard input

```
2 3
a 2 3
.*.
***
b 3 3
.*.
***
.*.
2 3
.*.
***
4 3
***
*.*
*.*
***
9 13
************.
*..........*.
*.......*..*.
*...*..***.*.
*..***.....*.
*...*......*.
*..........*.
************.
.............
```

standard output

```
a
[]
[ba]
```

Problem J. Ancient Scrolls

Input file:	standard input
Output file:	standard output
Time limit:	2 seconds
Memory limit:	512 mebibytes

You bought 3 ancient scrolls from a magician. These scrolls have a long string, and the lengths of the strings are the same. He said that these scrolls are copies of the key string to enter a dungeon with a secret treasure. However, he also said, they were copied so many times by hand, so the string will contain some errors, though the length seems correct.

Your job is to recover the original string from these strings. When finding the original string, you decided to use the following assumption.

- The copied string will contain at most d errors. In other words, the Hamming distance of the original string and the copied string is at most d.
- If there exist many candidates, the lexicographically minimum string is the original string.

Can you find the orignal string?

Input

The first line of the input file contains two integers l ($1 \leq l \leq 100{,}000$) and d ($0 \leq d \leq 5{,}000$.) l describes the length of 3 given strings and d describes acceptable maximal Hamming distance. The following 3 lines have given strings, whose lengths are l. These 3 strings consist of only lower and upper case alphabets.

Output

Print the lexicographically (by the ASCII codes of characters) minimum satisfying the condition in a line. If there do not exist such strings, print "-1".

Examples

standard input	standard output
3 1 ACM IBM ICM	ICM
5 2 iwzwz iziwi zwizi	iwiwi
1 0 A B C	-1
10 5 jLRNlNyGWx yyLnlyyGDA yLRnvyyGDA	AARAlNyGDA