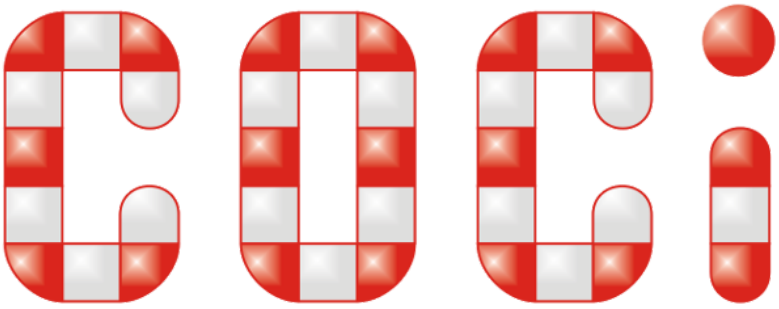


COCI 2017/2018

Round #3, November 25th, 2017

Tasks

Task	Time limit	Memory limit	Score
Aron	1 s	64 MB	50
Programiranje	3 s	64 MB	80
Retro	0.5 s	512 MB	100
Portal	1 s	256 MB	120
Dojave	4 s	256 MB	140
Sažetak	0.5 s	64 MB	160
Total			650

The holiday season is near! Aron wants to get gifts for his friends in Zagreb, so in order to get them on time, he visited a famous toy store in London. After picking out the gifts, he went to the register and discovered that there were already N people in line. Luckily, he noticed that there were groups of people standing in line, in addition to individual customers. A group of people consists of a customer and their friends waiting for them to complete the purchase. The moment when the customer is done, they and their friends leave the line.
The people considered a group are standing one behind the other and are wearing shirts of matching colour. Two adjacent groups, adjacent individuals or adjacent individual and a group, will never be wearing shirts of the same colour.

Write a program that will, given the data on the people standing in line, output which person in line Aron is.

INPUT

The first line of input contains the positive integer N ($1 \leq N \leq 25$) from the task.
Each of the following N lines contains a single character, an uppercase letter of the English alphabet that represents the shirt colour of the i^{th} person in line.

OUTPUT

You must output the required number from the task.

SAMPLE TESTS

input	input	input
3 C Z P	6 C C P C Z Z	6 B B B B B B
output	**output**	**output**
4	5	2

Clarification of the second test case:
First in line is the group consisting of two people in red shirts. Second in line is an individual in the blue shirt, third in line is an individual in the red shirt, and fourth in line is a group in green shirts. This makes Aron fifth in line.

Little Leticija is preparing for a programming exam. Even though she has solved a lot of tasks, there's one still left unsolved, so she is asking you for help. You are given the word S and Q queries. In each query, you are given positive integers A, B, C and D. Let's say that word X consists of letters between positions A and B in word S, and word Y from letters between positions C and D in word S. For each query, you must answer if it is possible to somehow rearrange the letters in word Y and obtain word X.

INPUT

The first line of input contains the word S ($1 \leq |S| \leq 50\,000$). $|S|$ denotes the number of characters in word S, which consists of lowercase letters of the English alphabet. The second line of input contains the positive integer Q ($1 \leq Q \leq 50\,000$).
Each of the following Q lines contains four integers A, B, C i D ($1 \leq A \leq B \leq |S|$ and $1 \leq C \leq D \leq |S|$) from the task.

OUTPUT

For each query, output "DA" (Croatian for yes) if it is possible, and "NE" (Croatian for no) if it is not.

SCORING

In test cases worth 50% of total points, it will hold: $1 \leq |S| \leq 1000$ and $1 \leq Q \leq 1000$.

SAMPLE TESTS

input	input	input
kileanimal 2 2 2 7 7 1 4 6 7	abababba 2 3 5 1 3 1 2 7 8	vodevovode 2 5 8 3 6 2 5 3 6
output	**output**	**output**
DA NE	DA DA	NE DA

Clarification of the third test case:
In the first query, X="vovo", and Y="devo". In the second query, X="odev", and Y="devo".

Little Mirko got a video game console for Christmas. It wasn't a *Playstation 4* or an *Xbox one*, but *Atari 2600*, and it came with one free game. The protagonist of the game is standing on the bottom of the screen, and there are various objects dispersed on the rest of the screen, falling towards the bottom.

More precisely, the screen can be represented as a grid of *R*x*S* pixels arranged in *R* rows and *S* columns. The protagonist takes up one pixel of the lowest line and is marked with 'M'. The rest of the pixels are marked with some of the characters: '.' (empty space), '*' (bomb), '(' (open bracket) or ')' (closed bracket).

The protagonist can move one pixel to the left or to the right in a single move, but doesn't need to, whereas the rest of the objects simultaneously move one pixel down (possibly out of the screen). When the protagonist finds himself at the same position as one of the brackets, we say that he picked up that bracket and added it at the end of his array of acquired brackets. The protagonist's goal is to acquire the longest possible **valid** bracket expression.

A valid bracket expression is defined inductively in the following way:

- "()" is a valid expression
- If **a** is a valid expression, then "(**a**)" is a valid expression as well
- If **a** and **b** are valid expressions, then "**ab**" is a valid expression as well

The game ends when the protagonist finds himself at the same position as the bomb, or when all the objects fall out of the screen.

INPUT

The first line of input contains the positive integers *R* and *S* ($1 \leq R, S \leq 300$) that represent the dimensions of the screen.
Each of the following *R* lines contains *S* characters 'M', '.', '*', '(' or ')' that represent the initial state of the screen.
Test data will be such that there will always exist at least one valid bracket expression that is possible to acquire.

OUTPUT

In the first line, you must output the length of the longest valid bracket expression that Mirko can acquire.
In the second line, output that expression. If there are multiple longest valid expressions, output the **lexicographically smallest** one.

SCORING

In test cases worth 25% of total points, it will hold $1 \leq R \leq 15$.
In test cases worth 50% of total points, it will hold $1 \leq R \leq 100$.

If you output the correct length, but the wrong expression, you will be awarded 40% of points for that test case. In any case, in order to score points, your output **must** consist of two non-empty lines.

SAMPLE TESTS

input	input	input
5 4 ..). .)(. (.)* *(.* ..M.	6 3)(. *.. (**)() (). M..	6 3 ((. *.. (**)() (). M..
output	**output**	**output**
4 (())	4 ()()	2 ()

Clarification of the first test case: The protagonist's moves are: left, left, right right.
Clarification of the second test case: The protagonist's moves are: stay still, stay still, stay still, right, left.
Clarification of the third test case: The protagonist's moves are: stay still, stay still, right.

The protagonist of this task, Chell, must solve a new puzzle GLaDOS has come up with. Chell is in a room whose layout that can be represented as a matrix of dimensions *N* rows and *M* columns. Each field can be one of the following:

- Obstructed field - there is a wall in it (denoted as '#'),
- The field where Chell is initially (denoted as 'C'),
- The field where Chell must get to in order to solve the puzzle (denoted as 'F'), or
- An empty field (denoted as '.').

Chell is carrying a so-called *portal gun*, a gun with which you can create portals in the walls. In each move, she can do one of the following:

- Move to an adjacent field using one move up, down, left or right (she cannot move to the field with a wall in it). This move lasts one unit of time.
- Create a portal in the wall by turning towards a wall, not necessarily an adjacent one, in the direction up, down, left or right and shooting. The portal will be created only on the side of the wall it was hit from. In each moment, **at most two portals can be active**. If a new portal is being created in the moment when two portals are already active, the one that was created earlier will disappear. It is not possible to create a new portal at the position of another existing portal. This move lasts a negligible amount of time, i.e. zero amounts of time.
- If she's at a field that is adjacent to a wall and there's a portal on her side of the wall, she can step into the portal and exit to a non-obstructed field with another portal. This move is possible if there are two active portals and lasts one unit of time.

Chell wants to know the minimal amount of time it takes for her to solve the puzzle, i.e. to reach the field denoted as 'F'.

Please note: The room will always have walls on the sides, and letters 'C' and 'F' will appear only once in the matrix.

INPUT

The first line of input contains the positive integers *N* and *M* ($4 \leq N, M \leq 500$), the numbers from the task.
Each of the following *N* lines contains *M* characters that describe the layout of the room.

OUTPUT

You must output the minimal amount of time it takes to solve the puzzle, or "nemoguce" (without quotation marks, Croatian for impossible) if it is not possible to solve it.

SCORING

In test cases worth 50% of total points, it will hold $4 \leq N, M \leq 15$.

SAMPLE TESTS

input	**input**	**input**
4 4 #### #.F# #C.# ####	6 8 ######## #.##..F# #C.##..# #..#...# #.....## ########	4 5 ##### #C#.# ###F# #####
output	**output**	**output**
2	4	nemoguce

Clarification of the second test case:

The puzzle can be solved in 8 moves, illustrated in the pictures below.

In the first move, we turn towards the left wall, shoot and create a portal that appears on the wall in the 3rd row and 1st column (coordinates (3,1)) from the right side.

In the second move, we create a portal from the upper side of the wall at coordinates (6,2).

In the third move, we step into the portal at coordinates (3,1) and exit at coordinates (5,2) - a non-obstructed field with the second portal.

In the fourth move, we turn right and create a portal from the left side of the wall at coordinates (5,7). Since there are already two portals, the one at field (3,1) disappears.

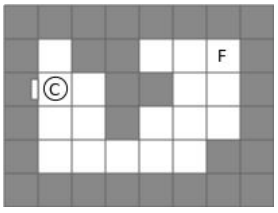
In the fifth move, we step into the portal at coordinates (6,2) and exit at coordinates (5,6) with the second portal.

In the sixth move, we create a new portal from the lower side of the wall at coordinates (1,6), making the portal at coordinates (6,2) disappear.

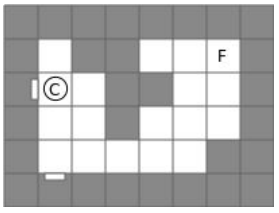
In the seventh move, we step into the portal at coordinates (5,7) and exit at coordinates (2,6).

Finally, in the eighth move, we move one place to the right to end the game.

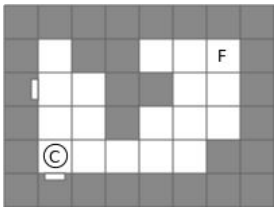
The portal creation in moves 1, 2, 4 and 6 lasts zero amounts of time, whereas the rest of the move last one unit of time, so the total time needed to solve the puzzle is 4 units of time.



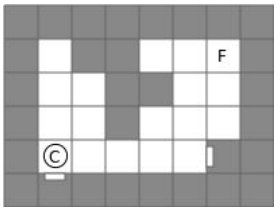
Move 1



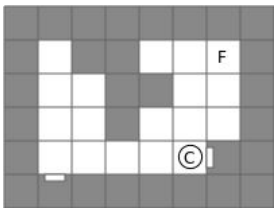
Move 2



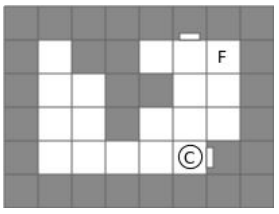
Move 3



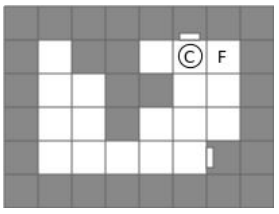
Move 4



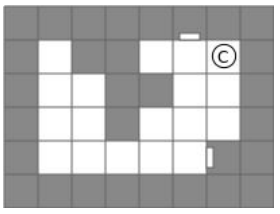
Move 5



Move 6



Move 7


Move 8

The biggest event of the year ended tragically for Croatian teams. The most influential theoretician of CERC of all time, the founder of the popular page *CERC Tips*, and in his free time an outstanding bass player, in his most recent performance failed to get his team to the finals.

In order to get over his existential troubles, our subject is spending time playing games of chance. He is especially interested in the following game:

You are given a positive integer M. Our protagonist sees in front of him a **permutation** of an array of numbers 0, 1, 2, ..., $2^M - 1$**.**
The computer chooses a **nonempty contiguous subsequence** of the given permutation, which it then lights up over a capital city of one of the countries in Southeastern Europe.
Our confidant, after fighting off tears caused by memories of old times, **must** choose two distinct elements of the permutation and **swap their places**. Our man of the hour wins if and only if the **bitwise XOR** of the numbers in the lit up subsequence after the substitution is **precisely** $2^M - 1$.

Our hero wants to know **the number of contiguous subsequences** the computer can light up so that he can win.

Help our hero overcome his (id)entity crisis so our favourite page can be fully active again.

INPUT

The first line of input contains the integer M ($1 \leq M \leq 20$),
The following line contains 2^M space-separated numbers that make up a permutation of the array 0, 1, 2, ..., $2^M - 1$.

OUTPUT

You must output the total number of contiguous subsequences that a computer can light up so our hero can win.

SCORING

In test cases worth 50% of total points, it will hold $1 \leq M \leq 14$.

SAMPLE TESTS

input	input	input
2 0 1 2 3	3 3 7 0 4 6 1 5 2	4 13 0 15 12 4 8 7 3 11 14 6 10 1 5 9 2
output	**output**	**output**
9	33	133

Clarification of the test cases:
In the first test case, if the computer chooses the subsequence [1 2 3], our hero can replace the numbers 0 and 3. In this case, he can actually win for every chosen contiguous subsequence, except the entire array.

In the second test case, if the computer chooses the entire array [3 7 0 4 6 1 5 2] as the lit up subsequence, our hero can't change the XOR of the subsequence (which is 0), no matter which two elements are swapped.

An unknown array *x* consists of N integers. The *K-summary* of that array is obtained by dividing the array into segments of length K and summing up the elements in each segment. If N is not divisible by K, the last segment of the division will have less than K elements.

In other words, the K-summary is an array where the elements are, respectively: (*x*[1] + … + *x*[K]), (*x*[K+1] + … + *x*[2K]), and so on, where the last sum that contains *x*[N] can have less than K summands. For example, the 5-summary of an array of 13 elements has three elements (sum of elements 1.-5., sum of elements 6.-10., sum of elements 11.-13.).

It is clear that we cannot reconstruct the elements of the original array from the K-summary, but that might be possible if we knew several K-summaries for different Ks. Write a program that will, given length N and set K_1, K_2, …, K_M, predict how many elements of the original array we would be able to uniquely determine if we knew all the K_i-summaries of the array. (It is not difficult to show that the number of reconstructed elements is independent of the content of the summaries.)

INPUT

The first line contains the integers *N* and *M* ($3 <= N <= 10^9$, $1 <= M <= 10$), the array length and the number of K-summaries.
The second line contains distinct integers K_1, K_2, …, K_M ($2 <= K_i < N$) from the task.

OUTPUT

You must output the required number of reconstructed elements.

SCORING

In test cases worth 40% of total points, it will hold N <= 5 000 000.

SAMPLE TESTS

input	input	input
3 1 2	6 2 2 3	123456789 3 5 6 9
output	**output**	**output**
1	2	10973937

Clarification of the first example: We can determine one element: x[3].
Clarification of the second example: We can determine x[3] and x[4].