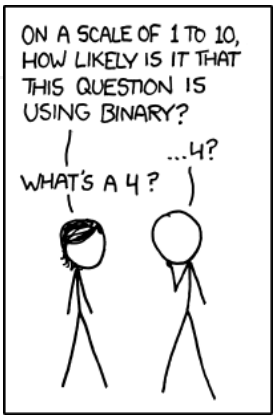
Binary Zip

Description



Two numbers "zip" together if their binary representations line up such that the 0's and the 1's are all opposite.

The unsigned 8 bit integer 170 is represented as 10101010. the 8 bit unsigned integer that will "zip" with 170 is 85, which in binary is 01010101.

```
10101010
01010101
```

Input

The input will consist of two lines. The first line will have a number N (1 <= N <= 32), the number of bits in each number. The second line will contain 2 N-bit unsigned integers A and B separated by a space.

Output

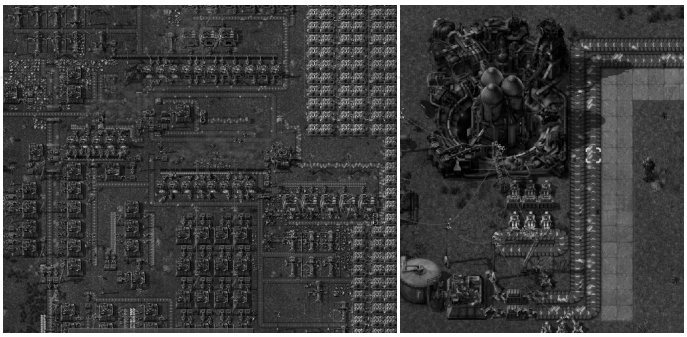
If A and B make a binary zip, print `YES`. Otherwise, print `NO`.

Examples

Input	Input	Input	Input	Input
1 0 1	8 170 85	5 13 18	5 5 27	8 13 18
Output	**Output**	**Output**	**Output**	**Output**
YES	YES	YES	NO	NO

Conveyor Loops

Description



Factorio is a game in which the player must build an elaborate network of machinery and conveyor belts to produce increasingly complex goods until they finally build a rocket. Players quickly find that loops of conveyor belts can be extremely useful.

You are presented with a grid of 1m x 1m squares, each being empty or having a single conveyor belt.

Each conveyor belt moves resources UP, LEFT, DOWN, or RIGHT; this direction does not change. All conveyor belts move at the same speed. Empty spaces do not move resources.

Find the longest loop of conveyor belts. In other words, determine the longest path a resource could take before returning to its starting position.

Input

The first is two space-separated integers r and c ($1 \leq r, c \leq 10{,}000$), the number of rows and columns in the conveyor belt grid, respectively. The next r lines of input each consist of c space-separated characters: `^` (UP), `>` (RIGHT), `<` (LEFT), `v` (DOWN), or `-` (EMPTY).

Output

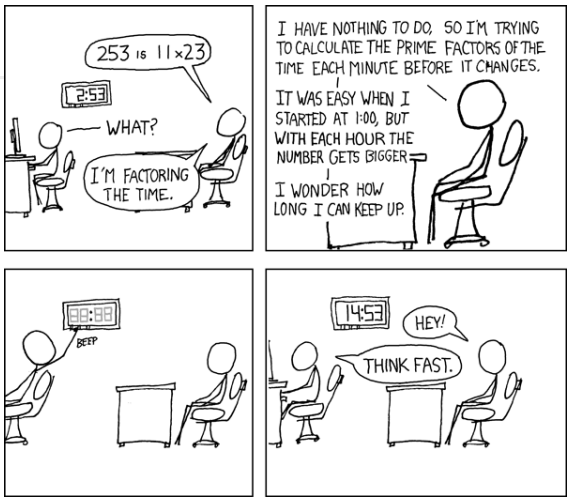
Output the length of the largest loop of conveyor belts. If no loops exist, output 0.

Examples

Input	Input	Input
1 2 > <	5 5 > > > > v ^ - v - v ^ - ^ - v ^ < < - v ^ < < < <	5 5 > > > > v ^ ^ - - v - - - - > v v - - ^ > > > > >
Output	**Output**	**Output**
2	16	0

No squares2

Description



Factoring a positive integer is decomposing it into a product of one or more integers greater than 1. For example, 24 can be factored as 24, 2×12, 3×8, 4×6, 2×2×6, 2×3×4, or 2×2×2×3.

A **square** is an integer that can be factored as two equal integers. For example, 4 is square because it can be factored as 2×2.

If a factoring has any factor that is divisible by a square greater than 1, let's call it **squarish**. For example, since 4 is a square, five of the factorings of 24 are squarish: 24, 2×12, 3×8, 4×6, and 2×3×4. That leaves two non-squarish factorings: 2×2×6 and 2×2×2×3.

Given a postive integer, find the shortest possible length of a non-squarish factoring. For example, for 24, the shortest possible length is three (2×2×6).

Input

An integer between 2 and 2,000,000 inclusive.

Output

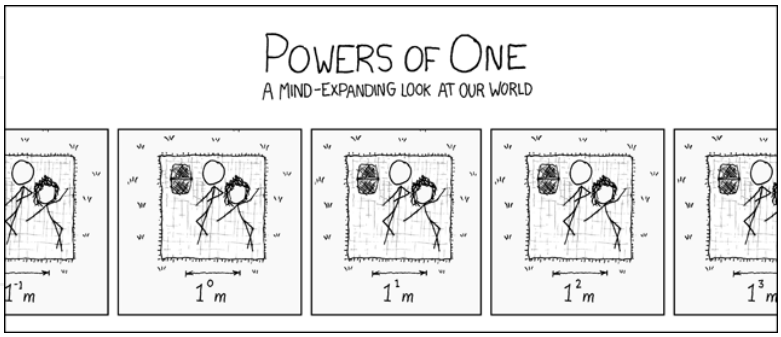
The shortest length of the integer's non-squarish factoring.

Examples

Input	Input
24	4
Output	**Output**
3	2

Power of N

Description



A number k is a power n if there exists an integer $i \geq 0$ for which $k = n^i$.

Input

The first line is k, and the second line is n. $1 \leq k, n < 2^{32}$.

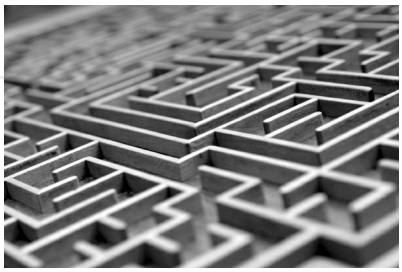
Output

Output `YES` if k is a power of n, or `NO` otherwise.

Input	Input
25 5	25 4
Output	**Output**
YES	NO

Rat race

Description



Your company, RatMaster Inc, market leader in rat maze fabrication has enlisted you to write a program to rate the difficulty of various maze configurations. One of the measures of difficulty they want you to report is the longest path in each maze. You must write a program which, given a series of maze layouts, analyzes each and computes the longest path that a rat can walk without back tracking.

Input

The input consists of T test cases. The quantity of them (T) is given on the first line of the input file. Each test case begins with a line containing two integers C and R ($3 \leq C,R \leq 1000$) indicating the number of columns and rows. Then exactly R lines follow, each containing C characters. These characters specify the labyrinth. Each of them is either a hash mark (#) or a period (.). Hash marks represent impassible walls, periods are free spaces where a rat can walk. Eat space has a unit length and width of 1.

The labyrinths are designed in such a way that there is exactly one path between any two free blocks.

Output

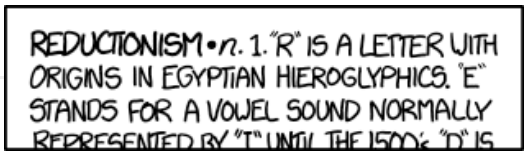
Your program must print exactly one line of output for each test case. The line must contain the sentence "Maximum path length is X." where X is the integer length of the longest path within the respective maze.

Examples

Input	Input	Input
1 4 4 #### #..# ##.# ####	2 3 3 ### #.# ### 6 6 ###### #....# #.##.# ####.# #....# ######	1 6 5 #.###. #..#.. ##.##. ..... #####.
Output	**Output**	**Output**
Maximum path length is 2	Maximum path length is 0 Maximum path length is 10	Maximum path length is 10

Reductionism

Description



As a programmer, you break big problems into smaller problems.

To prepare for reducing a word, your current task is to print each word with one letter per line.

Input

The input is one word, containing only lowercase letters (a - z).

Output

Print the word ready to be reduced, with one character on each line.

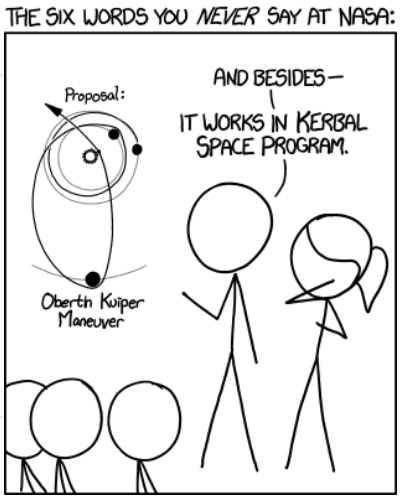
Examples

Input	Input	Input
`reductionism`	`programming`	`lucidchart`
Output	**Output**	**Output**

`r`
`e`
`d`
`u`
`c`
`t`
`i`
`o`
`n`
`i`
`s`
`m`
	`p`
	`r`
	`o`
	`g`
	`r`
	`a`
	`m`
	`m`
	`i`
	`n`
	`g`
		`l`
		`u`
		`c`
		`i`
		`d`
		`c`
		`h`
		`a`
		`r`
		`t`

Near miss

Description



We are ready for the Orberth Kuiper Maneuver, except that we have just detected a comet near the beginning of the planned route.

Find the shortest distance between the rocket and the comet after launch. For the purposes of this calculation, we'll approximate the trajectories of the rocket and the comet as straight lines with constant velocities.

Input

The first line is the starting 3D position of the rocket, given in meters as three integers between -10^6 and 10^6 inclusive. The second line is the 3D velocity of the rocket, given in meters per second as three integers between -10^3 and 10^3 inclusive. The next two lines are the same as the first two, except they describe the position and velocity of the comet.

Output

Output the shortest distance in meters that the rocket will be from the comet, beginning now. You may output any precision, as long as the value is accurate to within 0.001.

Input	Input

`0 0 0`
`0 1 0`
`1000 1000 1000`
`1 0 0`
	`10 10 10`
	`-1 -1 -1`
	`100 100 100`
	`1 1 1`

Output	**Output**
`500`	`155.885`

Swiss Bank Account

Description

Excentric multi-millionaire Thurston Howell III died in 1973. His vast fortune was held in a swiss bank account and his only heir, his son Gerald was never able to access the fortune because he could not find the account number amidst his father records. Gerald recently passed away and with his passing several teams of fortune hunters have begun searching for the secret account number hoping to be the first to claim Thurston's fortune. You have been contacted by a group who claims to hold the secret to finding the account number and are asking for your help. They have sent you a picture of a single page that appears to contain the specifications for a primitive (by modern standards) assembly language. This group claims to have possesion of a program that, when run, would output the bank account number. Your job is to implement a VM to run specified language since no existing processor exists that can run this assembly language.

Specification

```
REGISTERS:
    IP - Instruction Pointer, points to the next instruction to be executed, 000 at start
    R00, R01, R02, ..., R08 - 8 bit registers, initialized to zero at start
    CR - stored result of last CMP operation, can be >, =, <, initialized to = at start

INSTRUCTION FORMAT:
Each line of code will be formatted like:
NNN III OP1 OP2
Where NNN is the instruction number, starting at 000 and rising incrementally for each line
      III is a three letter instruction code, see below for instruction codes
      OP1 is the first parameter to the instruction, some instructions do not require this
      OP2 is the second parameter to the instruction, some instructions do not require this

INSTRUCTIONS:
   SET OP1 OP2 - load constant value specified by OP2 into register specified by OP1
   ADD OP1 OP2 - add value in register specified by OP2 to register specified by OP1,
                 store result in register OP1
   CMP OP1 OP2 - compare value in register OP1 with value in register OP2, stores result
                 in CR
   JMP OP1     - set IP to instruction number OP1 which must be a constant value
   JIF OP1 OP2 - conditional jump.  OP1 must be one of GRE, LES, EQU, GRQ, LEQ NEQ where
                 GRE is >,  LES is <. EQU is =, GRQ is >=, LEQ is <= and NEQ is !=.
                 Compares CR to specified condition and executes a jump to line
                 OP2 if condition is met
   OUT OP1     - outputs a single character
   END         - end of the program, all programs must end with this instruction

Constant values can be in one of two forms:
   *  DDD where D is a numeral from 0-9.  0 -> 000, 37 -> 037
   *  'c' where c is any ascii character
```

Input

Your input will be several lines of assembly code as described in the specification.

Output

You will output what the assembly code would output via its OUT command.

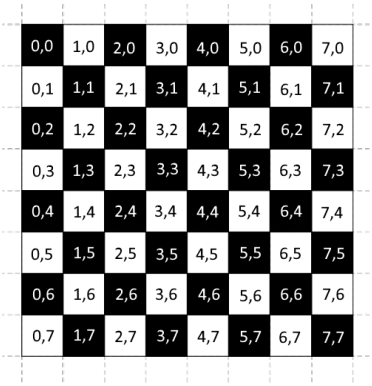
Examples

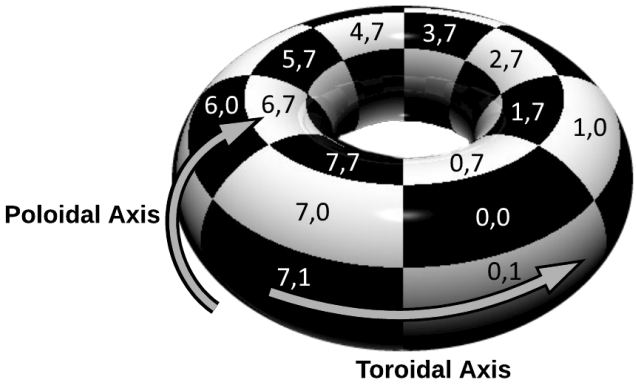
Input	Input	Input
000 SET R00 'H' 001 OUT R00 002 SET R00 'e' 003 OUT R00 004 SET R00 'l' 005 OUT R00 006 OUT R00 007 SET R00 'o' 008 OUT R00 009 END	000 SET R00 '0' 001 SET R01 001 002 SET R02 '9' 003 OUT R00 004 ADD R00 R01 005 CMP R00 R02 006 JIF LEQ 003 007 END	000 SET R00 13 001 SET R01 '0' 002 SET R02 '9' 003 SET R03 101 004 ADD R00 R03 005 CMP R00 R01 006 JIF LES 004 007 CMP R00 R02 008 JIF GRE 004 009 OUT R00 010 CMP R00 R01 011 JIF NEQ 004 012 END
Output	**Output**	**Output**
Hello	0123456789	5381649270

King's wrapped journey

Description

Imagine a chess board wrapped around a Torus as shown in the figure below.





This wrapping will cause top edge to be connection to the bottom edge of the chess board, as well as left edge to be connected to the right.

A King is a chess piece that may move to any directly adjacent or diagonal square in a single turn. Calculate the minimum number of turns required for a King to travel from a given starting position to a given destination.

Input

The first line is a number *N*. *N* testcases follow. Each test case consists of two space-separated coordinates. Each coordinate will appear on a separate line of input.

Output

Output a single integer for each test case indicating the minimum number of turns that are required for the king to move from the given starting coordinates to the given destination coordinates.

Examples

Input	Input
1 0 0 0 1	2 0 0 1 2 0 0 7 7
Output	**Output**
1	2 1

Word find

```
T Q J V I C T O R Y Q Q Q
Q P Q C Q Q C Q N E R D S
W R Q Z O Q O P K F A E E
E O C X N M M Q Q Q Q Q Q
R G V W N P P R O G R I M
T R Q Q M Q E E Q Q Q Q Q
Y A Q Q P Q T Q T Q V Q E
U M Q Q O T L Q N E R O S

COMPETE    NERDS
PROGRAM    VICTORY
```

Description

Solve a word search puzzle. Words may be found in four ways

- left to right: every character appears one column to the right of the preceding character.
- top to bottom: every character appears one row below the preceding character.
- up diagonal: every character appears one row above and one character to the right of the preceding character
- down diagonal: every character appears one row below and one character to the right of the preceding character

Input

The first line consists of two space-separated integers *r* and *c*. The next *r* lines each have *c* space-separated characters, representing the word search grid. The next line is *n*. The next *n* lines are the words to find.

Output

Output the word search grid as it was given, but replace any letters not part of a match with a period.

Examples

Input	Input
5 5 Q X Q X Q Q A C M Q Q R A R Q Q Q R Q Q T E S T P 3 ACM CARS TEST	3 9 A Z S D F Q Y R W N C P B N E T W U G E M E H Z X R T 3 ACM HEY WUT
Output	**Output**

```
. . . . .
. A C M .
. . A . .
. . R . .
T E S T .
```

```
A . . . . . Y . W
. C . . . E . . U
. . M . H . . . T
```