Problem A. Football Scoring

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

There are five ways to score points in american professional football:

1. Touchdown - 6 points
2. Field Goal - 3 points
3. Safety - 2 points
4. After touchdown
   (a) 1 point (Field Goal or Safety) - Typically called the “Point after”
   (b) 2 points (touchdown) - Typically called the “Two-point conversion”

(https://operations.nfl.com/the-rules/nfl-video-rulebook/scoring-plays/)

Given the box score values for two competing teams, calculate the point total for each team.

Input

There are two lines of input each containing five space-separated non-negative integers, T, F, S, P and C representing the number of **T**ouchdowns, **F**ield goals, **S**afeties, **P**oints-after-touchdown and two-point **C**onversions after touchdown respectively. $(0 \leq T \leq 10)$, $(0 \leq F \leq 10)$, $(0 \leq S \leq 10)$, $(0 \leq (P+C) \leq T)$. The first line represents the box score for the visiting team, and the second line represents the box score for the home team.

Output

The single output line consists of two space-separated integers showing the visiting score and the home score respectively.

Example

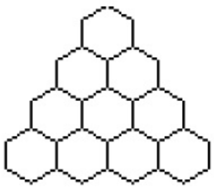
Input	Output

`1 3 0 0 1`
`3 1 1 1 1`
	`17 26`

Problem B. Tri-Color Puzzle

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

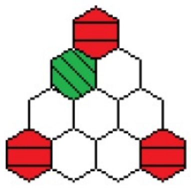
A **Tri-Color Puzzle** is a triangular array of hexagon cells with S cells on each side for a total of $N = \frac{S \cdot (S+1)}{2}$ cells. For example, the following is a puzzle with side 4 and 10 cells.

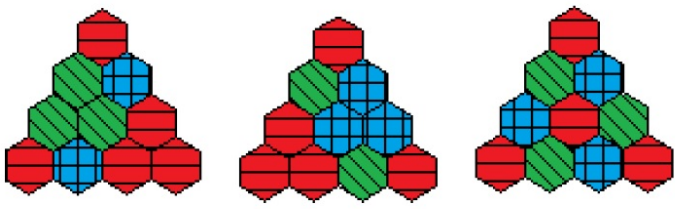


To solve the puzzle, each cell must be colored red, green or blue so that for each triplet of cells with one cell above and two cells below, either all of the three cells are the same color or each of the three cells is a different color. For the purposes of this problem statement, we will also use hash patterns as well as colors for clarity as follows:

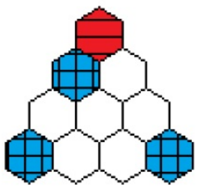
Red = 🟥 Green = 🟩 Blue = 🟦

In a particular puzzle, some of the cells will be initially specified and the remaining cells are to be filled in as described above. The following example has three solutions:

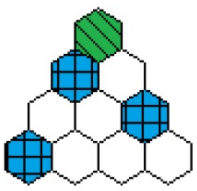


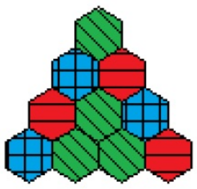


This example has no solutions:



This example has exactly one solution:





Write a program which takes as input a description of a **Tri-Color puzzle** and outputs the number of solutions to the puzzle.

Input

Input consists of multiple lines of input. The first line contains 2 space separated decimal integers, S and I, $(3 \leq S \leq 19)$ and $(0 \leq I \leq N = \frac{S\cdot(S+1)}{2})$, The first line is followed by I lines of three decimal integers giving the row, r, the cell in the row, c, and the color code cc for one initial cell color $(1 \leq r \leq S)$, $(1 \leq c \leq r)$ and $(0 \leq cc \leq 2)$ where $0 \implies$ red, $1 \implies$ green and $2 \implies$ blue.

Output

The single output line consists of the number of solutions to the **Tri-color puzzle** specified by the input.

Example

Input	Output

4 4
1 1 0
2 1 2
4 1 2
4 4 2
	0

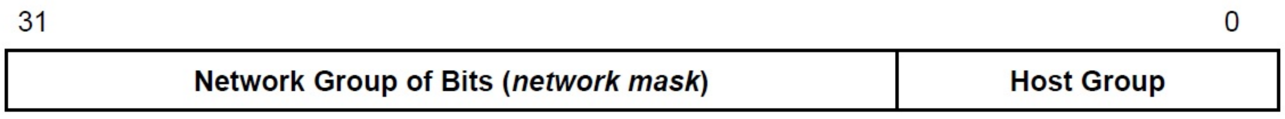
4 4
1 1 1
2 1 2
4 1 2
3 3 2
	1

4 4
1 1 0
2 1 1
4 1 0
4 4 0
	3

Problem C. CIDR

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

Internet Protocol (IP) V4 addresses are 32-bit integers that uniquely identify a host on the internet. IP addresses are written in a *dotted-quad* notation such as 192.168.1.100. Each component of the dotted-quad specifies an 8-bit (unsigned) value, corresponding to each of the 4 bytes in the 32-bit integer address. An IP address specifies two groups of bits in that 32-bit integer: the network group and the host group. The least significant bits in the address specify the specific host on the network described by the most significant bits.



The *network mask* is a bitmask that has all the bits in the network group set to 1. Ex. If the *network group* has 14 bits, then the *netmask* is: `0xfffc0000` or, in *dotted-quad* format: `255.252.0.0`

Historically, each group consisted of a multiple of 8 bits: 8, 16 or 24. This led to problems with IP address allocation: the total number of networks were usually too big or too small for most organizations to use. The smallest number of hosts on a network was 256 (8 bits of host). This turns out to be a waste of IP address space, since rarely did small networks need 256 hosts.

The Classless Inter-Domain Routing (CIDR) was introduced to deal with this. It allows for any number of bits to be used for the network and host groups, as long as the total bits used by the network and host groups totals 32. CIDR notation specifies a *dotted-quad* IP Address, a slash ('/') character, and a decimal number (0-32). For example, `192.168.1.100/14` represents the IP address `192.168.1.100` and its associated network prefix of `192.128.0.0`, or equivalently a netmask of `255.252.0.0` which has 14 leading 1-bits.

For this problem, you will write a program that accepts a list of *dotted-quad* IP V4 addresses and determines the network group that contains all the specified IP addresses with the smallest number of unused or wasted hosts.

Input

The first line of input contains a single decimal integer N, $(2 \leq N \leq 65535)$, which is the number of *dotted-quad* IP addresses the follow. This is followed by N lines, each containing a valid *dotted-quad* IP address.

Output

The single output line consists of a single integer value representing the number of bits in the network group. That is, the integer value that follows the slash, ('/'), character in the CIDR representation of an address in the network group. If there is no common network group shared by the IP addresses, you should print 32, this is a special case which is commonly used to refer to specific hosts, that is, there is no network component. This implies you should never output a value of 0.

Example

Input	Output
4 10.0.0.1 10.0.0.24 10.0.0.8 10.0.0.16	27

Problem D. Sequinary Numerals

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

A sequinary numeral is a sequence of digits:

$$d_n d_{n-1} \dots d_1 d_0$$

where d_n is 1 or 2 and the others are 0, 1, or 2.

It represents the rational number:

$$d_0 + d_1 \cdot \left(\frac{3}{2}\right) + d_2 \cdot \left(\frac{3}{2}\right)^2 + \dots + d_n \cdot \left(\frac{3}{2}\right)^n$$

Write a program which takes **a sequinary numeral** as input and returns the number it represents as a proper fraction.

Input

The single line of input contains a sequinary numeral of no more than 32 digits

Output

Output consists of a single line.

If the result is an integer, the output is the decimal integer. Otherwise, the output is N a single space and K/M where N, K and M are decimal integers where $K < M$ and K/M is in lowest terms ($\text{GCD}(K, M) = 1$).

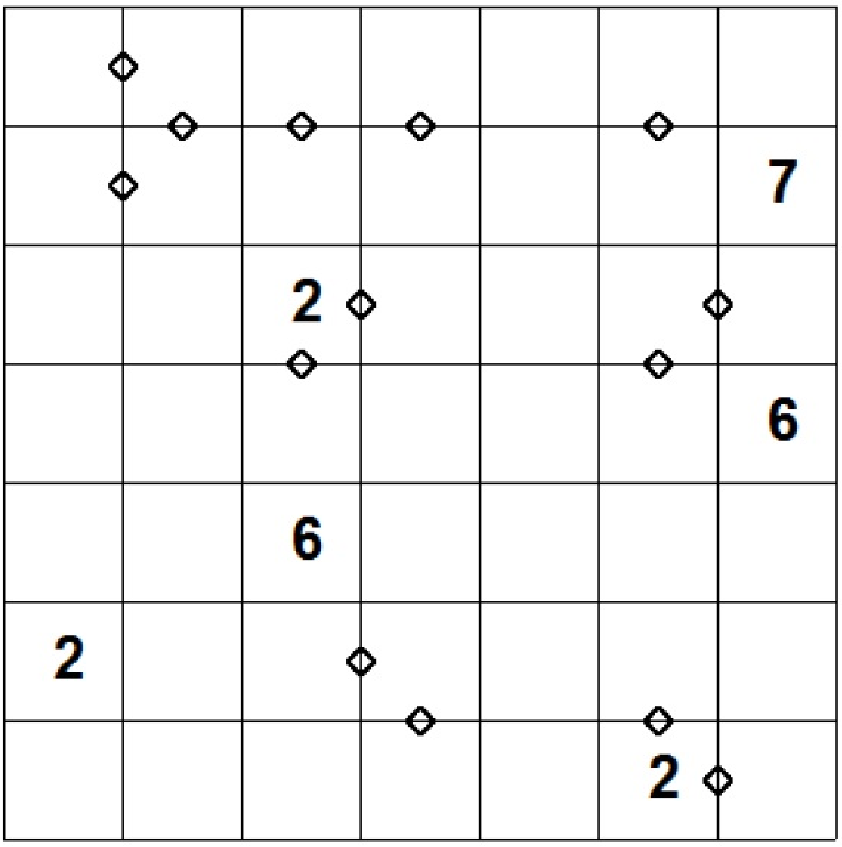
Example

Input	Output
`2101`	`10`
`201`	`5 1/2`

Problem E. Neighbors

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

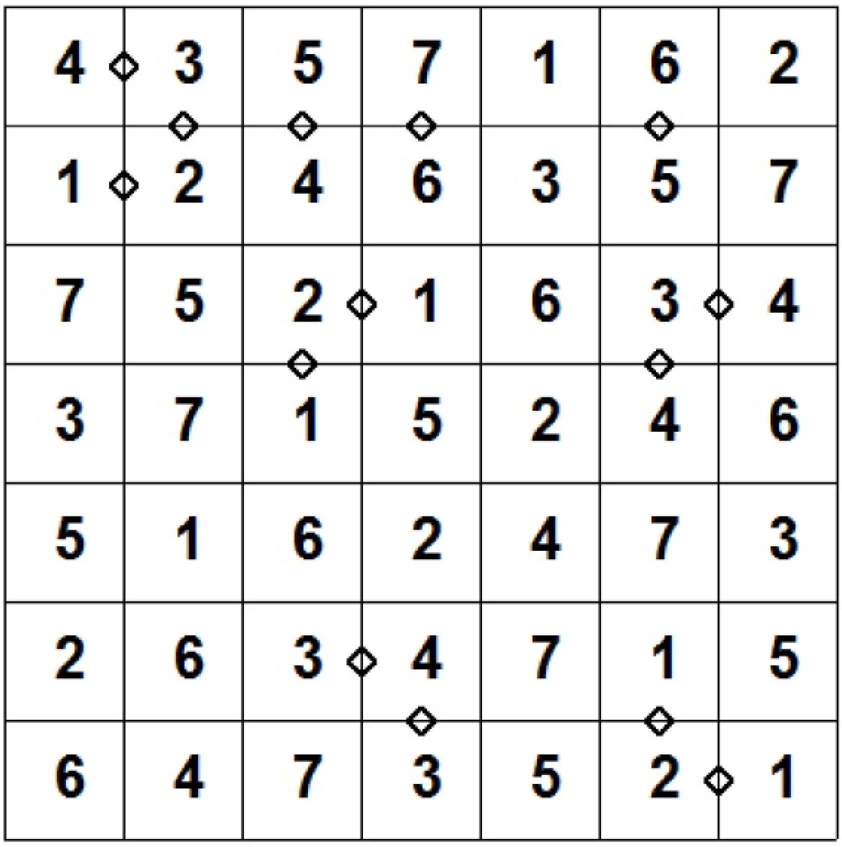
The **Neighbors Puzzle** is based on the idea that two integers are neighbors if they differ by one. The puzzle consists of a grid of N rows and N columns. On some of the internal edges are diamonds. In addition, a small number of values will be pre-specified (the 7 in row 2 column 7, for example).



To solve the puzzle, fill in the empty squares with integers from 1 to N, so that:

- In each row, each value from 1 to N appears exactly once.
- In each column, each value from 1 to N appears exactly once.
- If there is a diamond between two values, they are neighbors (differ by 1).
- If there is not a diamond between two values, they are not neighbors (differ by more than 1).

For example, a solution to the puzzle above is on the next page...



Write a program to solve Neighbor Puzzles.

Input

The first line of input contains two space separated decimal integers N, $(4 \leq N \leq 12)$ which is the number of rows and columns and K, $\left(\frac{N}{2} + \frac{N^2}{16} \leq K \leq N^2\right)$, which is the number of pre-specified values.

The next $2N-1$ lines of input consist of the values 0 or 1 indicating "*not a neighbor*" or "*is a neighbor*" respectively with no spaces between them.

The odd numbered rows of the set contain $N-1$ values corresponding to constraints on values on either side of vertical lines within a box.

The even numbered rows contain N values corresponding to constraints on the values above and below the symbol.

These $2N-1$ lines are followed by K lines of three space separated decimal integers. The values give the row, column and value in that order (all $1 \ldots N$) of each pre-specified value.

The input data supplied is guaranteed to generate a single unique solution to the puzzle.

Output

Your program should produce N lines of output where each line consists of N decimal digits separated by a single space. The value in the j-th position in the i-th line of the N output lines is the solution value in column j of row i.

Example

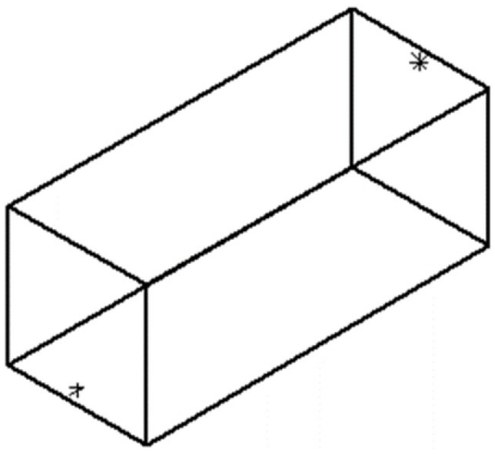
Input	Output
7 6	4 3 5 7 1 6 2
100000	1 2 4 6 3 5 7
0111010	7 5 2 1 6 3 4
100000	3 7 1 5 2 4 6
0000000	5 1 6 2 4 7 3
001001	2 6 3 4 7 1 5
0010010	6 4 7 3 5 2 1
000000	
0000000	
000000	
0000000	
001000	
0001010	
000001	
2 7 7	
6 1 2	
3 3 2	
7 6 2	
5 3 6	
4 7 6	

Problem F. Spider-Fly

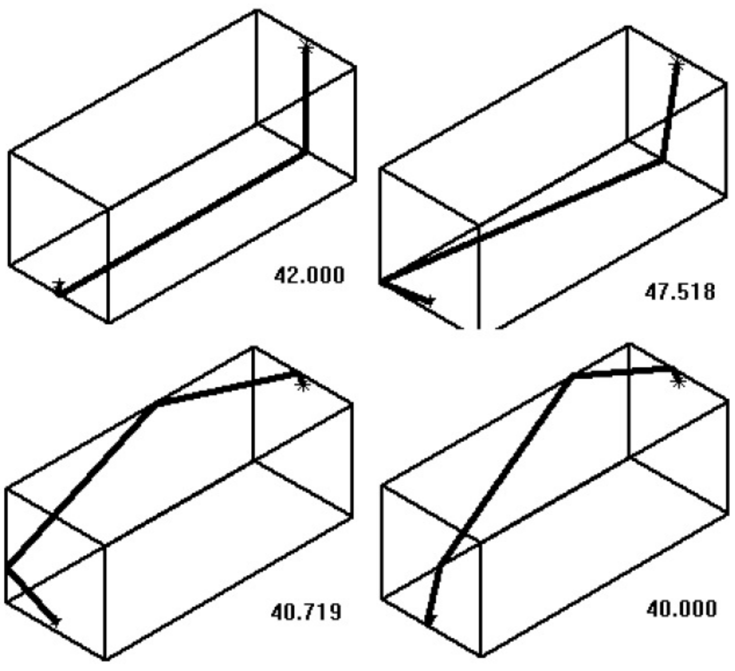
Source file name:	Spider.c, Spider.cpp, Spider.java, Spider.py
Input:	`Standard`
Output:	`Standard`

In the English newspaper *Weekly Dispatch* on 14 June 1903, Henry Dudeny posed the following problem (that's right, everyone is now typing "henry dudeny spider fly" into Google search).

In a room 12 feet wide, 12 feet high and 30 feet long, there is a spider 1 foot from the ceiling and horizontally centered on one end wall. There is a fly 1 foot from the floor and horizontally centered on the opposite wall. What is the shortest path (in feet) the spider can walk across walls, ceiling or floor to reach the fly?



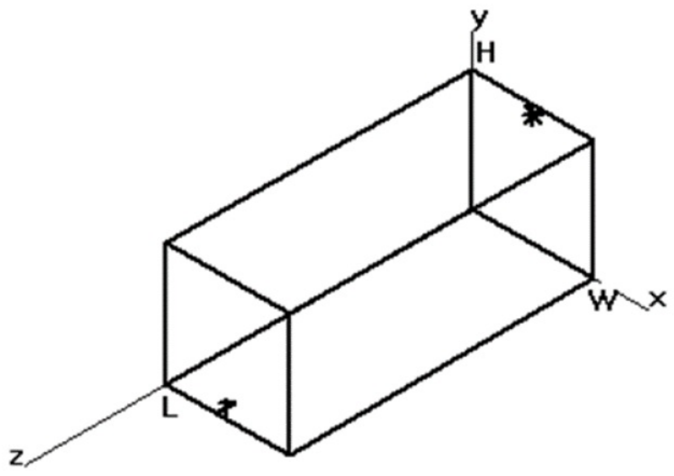
Several possible paths are shown below together with their lengths. The 40 foot path is (one of) the shortest.



Write a program which solves the general spider-fly problem.

Input

The input will use a coordinate system with origin at a corner of the room. The x-axis in the direction of the width, the y-axis in the direction of the height and the z-axis in the direction of the length so that $0 \leq x \leq Width$, $0 \leq y \leq Height$ and $0 \leq z \leq Length$:



The input consists of three lines of input. The first input line contains three double precision decimal numbers giving the *Width*, *Height* and *Length* (in feet) in that order ($0 < Width$, $Height$, $Length \leq 30$). The second line consists of three double precision decimal numbers: x, y and z, in that order, giving the coordinates of the spider. The third line consists of three double precision decimal numbers: x, y and z, in that order, giving the coordinates of the fly. In each of the latter two lines, the points will be on a wall, the floor or the ceiling, that is: either $x = 0$ OR $x = Width$ OR $y = 0$ OR $y = Height$ OR $z = 0$ OR $z = Length$.

Output

The single output line consists of a single floating point decimal value, accurate to three decimal places, giving the length of the shortest path in feet.

Example

Input	Output

12 12 30
6 11 0
6 1 30
	40.000

Problem G. Simple Collatz Sequence

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

The *Simple Collatz Sequence* (SCS) starting at an integer n, is defined by the formula:

$$S(k) = (k/2 \text{ if } k \text{ is even, else } (k+1))$$

The sequence is then n, $S(n)$, $S(S(n))$, $\ldots$ until the value first reaches 1.

For example, starting at 11, we have:

$$11 \to 12 \to 6 \to 3 \to 4 \to 2 \to 1$$

The sequence always ends at 1. (Fun Fact: The *Hard Collatz Sequence* sends odd k to $3 \cdot k + 1$. It is unknown whether that sequence always ends at 1.)

Let $A(n) =$ *number of steps in the SCS starting at* n. For example, $A(11) = 6$.

Let $C(n) =$ *the number of integers* m *for which* $A(m) = n$. For example, the integers for which $A(n) = 6$ are:

$$10,\ 11,\ 13,\ 24,\ 28,\ 30,\ 31,\ 64$$

So $C(6) = 8$.

Note that if $n > 2^m$, then $A(n) > m$ since we need to divide by 2 at least $m + 1$ times.

Write a program to compute $C(m)$.

Input

Input consists of a single line which contains a decimal integer, m, $(1 \le m \le 40000)$, which is the value for which $C(m)$ is to be found.

Output

The output consists of a single line that contains the value of $C(m)$ modulo 1000007.

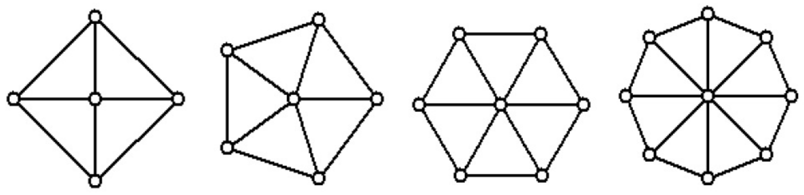
Example

Input	Output
6	8
12345	540591

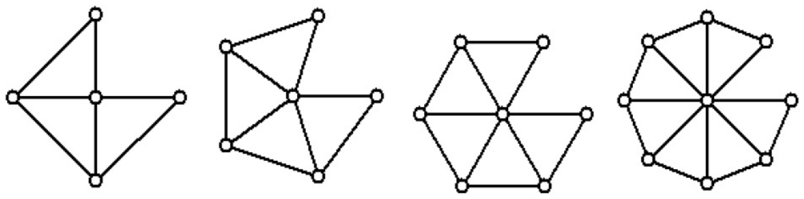
Problem H. How Many Unicycles in a Broken Wheel

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

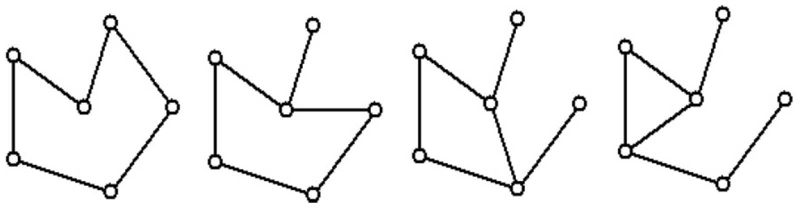
A **Wheel Graph** of size n is a cycle of n vertices, $v[1]$, $v[2]$, …, $v[n]$ each of which is connected to a center vertex, $v[0]$. Examples of wheel graphs of size 4, 5, 6 and 8 are shown below:



A **Broken Wheel Graph** of size n is a wheel graph of size n with the edge from $v[n]$ to $v[1]$ removed. Examples of broken wheel graphs of size 4, 5, 6 and 8 are shown below:



A **spanning unicycle** in a graph, G, is a spanning tree in G with one additional edge added to form a single cycle. Each of the examples below is a spanning unicycle in a broken wheel graph of size 5:



Write a program to compute the number of different unicycles in a broken wheel graph of size n. Recall that two subgraphs, $S1$ and $S2$, of a graph G are different if there is at least one edge of G that is in $S1$ and not in $S2$ OR an edge in $S2$ which is not in $S1$.

Input

Input consists of a single line that contains a decimal integer, m ($3 \leq m \leq 4000$), which is the size of the wheel graph to find the number of unicycles of.

Output

The single output line consists of the count of unicycles modulo 100007.

Example

Input	Output
5	19
1234	50380

Problem I. Pooling PCR Tests

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

The *Polymerase chain reaction* (PCR) test is a test for COVID which repeatedly copies a DNA sample and then tests for the presence of COVID specific DNA segments. This can take time and the required reagents may be limited. One suggestion for improving throughput is pooling of samples.

The idea is to combine N samples and run the PCR test on the combined sample. If the test is negative then no further test is required. If the test is positive, all N people must be retested individually. If the probability of a positive test is low, this will significantly reduce the number of tests required.

Write a program which takes as input the probability, p, that a single person tests positive and outputs the optimum number of samples, N, to combine.

If P is the probability that all N people test negative then the expected number of tests, $E(T)$, required is:

$$E(T) = 1 \cdot P + N \cdot (1 - P)$$

Choose N to minimize $E(T)/N$.

For example, if N is 2 and p is 0.5, P is 0.25 and $E(T) = 0.25 + 2 \cdot 0.75 = 1.75$ which is only slightly less than N.

In order to be sure that the sample from each person is sufficient, N must be no more than 16.

Input

Input consists of a single line containing a single decimal floating point value p, $(0 < p < 1)$, the probability that a single person tests positive for Covid.

Output

Output is a single line containing a single decimal integer. If $E(T) \geq N$ for all N, output the value 1. Otherwise, the value is N, $2 \leq N \leq 16$, which minimizes $E(T)/N$.

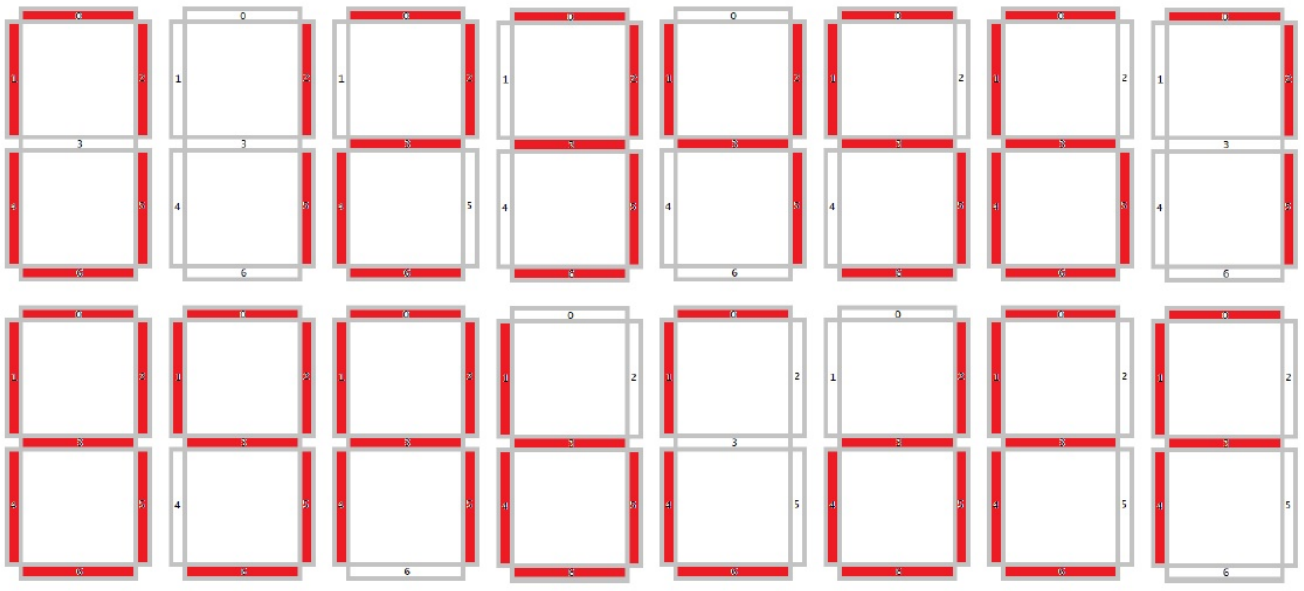
Example

Input	Output
0.1	4
0.02	8
0.01	10

Problem J. 7 Segments You Say?

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

A seven segment LCD display provides a mechanism to display a hexadecimal digit by turning on some combination of segments to form the digits as shown below by the RED segments.



The image above represents the entire hexadecimal character set [`0-9`, `AbCdEF`] for this particular seven segment display. Note that the digits '`b`' and '`d`' appear as lower-case letters to avoid the ambiguity with the digit 8 and 0 respectively. A special character where no segments are on can be used as a space.

Each segment is numbered 0 through 6 as shown above by the small numbers in each segment. So, for example, the digit `3` has segments 0, 2, 3, 5 and 6 turned on. Each segment can be represented by 1 bit (0 if the segment is not on and 1 if the segment is on). Each digit is represented by 7 bits. The low order bit corresponds to segment 0 and the high order bit corresponds to segment 6. For the digit `3`, the bit string is "`1101101`". For this problem, you will read a string of bits and display the seven segment output representation scaled in size by some factor.

Input

Input consists of a single line containing the scaling factor S, $(1 \leq S \leq 8)$, followed by a single space, followed by a bit string consisting of 1 to 1000 zeroes and ones (0's and 1's). If the bit string length is not a multiple of 7, you must pad it on the left with 0's (zeroes). Every 7 bits represents a single digit to display. Invalid digits should be displayed using the special character.

Output

Output consists of multiple lines depending on the number of digits to be displayed and the scaling factor. Each digit to be displayed will consist of a display cell that is $S \cdot 7$ rows and $S \cdot 3$ columns. Each horizontal segment that is on shall be composed of S lines of $S \cdot 3$ "`X`" characters and each horizontal segment that is not on shall be composed of S lines of $S \cdot 3$ space characters. Each vertical segment that is on shall be composed of $S \cdot 2$ lines of S "`X`" characters vertically. Each vertical segment that is not on shall be composed of $S \cdot 2$ lines of S space characters vertically. Vertical segments on the same row should have S space character(s) between them. Multiple digit cells will be placed next to each other leaving $S \cdot 2$ spaces between cells. The maximum output line length is 80 columns. Any digit cell that will not completely fit on the output row should be moved to the next line of cells, leaving S blank lines between cells and no **extra** space at the beginning or ends of any rows. Spaces that are part of the seven segment display should always be displayed, even at the end (or beginning) of a line.

Part of this problem is formatting. As such, the output will be strictly checked and must follow the format rules precisely as stated above.

Note: In the examples below, the "**Example Output**" shows a vertical bar, `|`, to denote the end of each output line. This vertical bar should NOT be shown in the output generated by your program. It is here for illustrative purposes ONLY.

In some of the "**Example Input**", it appears that each line is terminated by a single new-line character. In some examples, the single line may not fit in the "**Example Input**" box, so it is broken into multiple lines. Keep in mind, there are no new-lines in there, only the single new-line at the end of the line of input.

Example

Input

```
1 110111111011
```

Output

```
XXX  XXX|
X    X  |
X    X  |
XXX  XXX|
X    X X|
X    X X|
     XXX|
```

Input

```
1
1110111010010010111011101101010111011010111111011010010111111111101111011111111
1010101001111111001011011001101111101110100100101110111011010101110110101111101
1010010111111111101111011111111110101010011111110010110110011011111011101001001 0
111011101101
```

Output

```
XXX       XXX  XXX       XXX  XXX  XXX  XXX  XXX  XXX       XXX       XXX  XXX|
X X    X    X    X  X X  X    X      X  X X  X X  X X  X    X      X  X    X  |
X X    X    X    X  X X  X    X      X  X X  X X  X X  X    X      X  X    X  |
          XXX  XXX  XXX  XXX  XXX       XXX  XXX  XXX  XXX       XXX  XXX  XXX|
X X    X  X      X    X    X  X X    X  X X    X  X X  X X  X    X X  X    X  |
X X    X  X      X    X    X  X X    X  X X    X  X X  X X  X    X X  X    X  |
XXX       XXX  XXX       XXX  XXX       XXX  XXX       XXX  XXX  XXX  XXX     |
|
XXX       XXX  XXX       XXX  XXX  XXX  XXX  XXX  XXX       XXX       XXX  XXX|
X X    X    X    X  X X  X    X      X  X X  X X  X X  X    X      X  X    X  |
X X    X    X    X  X X  X    X      X  X X  X X  X X  X    X      X  X    X  |
          XXX  XXX  XXX  XXX  XXX       XXX  XXX  XXX  XXX       XXX  XXX  XXX|
X X    X  X      X    X    X  X X    X  X X    X  X X  X X  X    X X  X    X  |
X X    X  X      X    X    X  X X    X  X X    X  X X  X X  X    X X  X    X  |
XXX       XXX  XXX       XXX  XXX       XXX  XXX       XXX  XXX  XXX  XXX     |
|
XXX       XXX  XXX|
X X    X    X    X|
X X    X    X    X|
          XXX  XXX|
X X    X  X      X|
X X    X  X      X|
XXX       XXX  XXX|
```

Input

```
4 110111111011
```

Output

```
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXX                XXXX        |
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXXXXXXXXXX        XXXXXXXXXXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
XXXX                XXXX    XXXX|
                    XXXXXXXXXXXX|
                    XXXXXXXXXXXX|
                    XXXXXXXXXXXX|
                    XXXXXXXXXXXX|
```

Problem K. Transporting Spaghetti

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

The unique location of Venice in northeastern Italy poses a shipping problem in many cases due to the canals and whatnot. To transport large quantities of spaghetti from Milan to Venice, a company uses trucks capable of carrying A tons from Milan to Mestre, the city on the mainland closest to Venice, and boats capable of carrying B tons from Mestre to Venice. One day, the depot in Venice requests an arbitrary amount of spaghetti, but not less than C tons and the depot in Mestre requests exactly D tons. Write a program to determine the *smallest* number of trucks to be sent from Milan to satisfy both orders such that every truck and boat used for the transport is loaded to capacity.

Input

There is a single line of input containing the four integers, A, B, C and D. $(0 < A \leq 100)$, $(0 < B \leq 20)$, $(0 \leq C \leq 100)$, $(0 \leq D \leq 100)$.

Output

The single output line consists of the sentence: `We need` t `trucks and` b `boats.` Where t is the number of trucks required and b is the number of boats required. If t or b is 1, then you should not pluralize the words "`truck`" or "`boat`" respectively. If there is no solution that meets the criteria, output: `No solution.`

Example

Input	Output
`31 13 50 28`	`We need 3 trucks and 5 boats.`
`100 20 30 10`	`No solution.`
`1 1 1 100`	`We need 101 trucks and 1 boat.`

Problem L. Letterle

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

A new game is played by giving the user up to seven guesses to figure out a *letterle* (pronounced: letter-el). A *letterle* consists of five letters (`A-Z`), in a specific order. After each guess, the user is given feedback as to the correctness of their guess. Feedback is provided as a five letter string containing only the following letters: `X`, `Y` and `G`.

`X` = letter in this position does not appear in the letterle

`Y` = letter in this position appears in the letterle, but it is not in the correct position

`G` = letter in this position is correct (good).

For this problem, you will write a program that generates the feedback for up to seven guesses for a supplied *letterle*.

Input

The first line of input contains a five character string of capital letters (`A-Z`) that are the *letterle*. The next seven lines contain guesses that the program must evaluate and generate feedback for the user.

Output

Output consists of one or more lines. For each guess, if the guess is correct, the output line is `WINNER` and the program should not process any more input. If the guess is incorrect, and this is the seventh guess, the output line is `LOSER`. Otherwise, the output line is a five character string where each position has one of `X`, `Y` or `G` (as described above). First, any position that a letter is correct, should have a `G`. Then, any position where a letter is not in the *letterle*, should contain an `X`. The remaining positions should contain a `Y` indicating the letter is correct, but in the wrong position.

Example

Input	Output
LIMIT ABCDE FGHIJ KLMNO PQRST LIMIT LEMIT LAMIT	XXXXX XXXGX XYGXX XXXXG WINNER