*We can all benefit by doing occasional "toy" programs, when artificial restrictions are set up, so that we are forced to push our abilities to the limit. … The art of tackling miniproblems with all our energy will sharpen our talents for the real problems.*

***Donald E. Knuth***

# Quarterfinal

# Central region of Russia

## Rybinsk, October 12-13-2011

A. Papa Carlo (64 Mb, 1 sec / test) .......... 2
B. ~~Not~~ so trivial problem (64 Mb, 1 sec / test) .......... 3
C. Sort by sum of digits (64 Mb, 1 sec / test) .......... 4
D. Roman numerals (64 Mb, 1 sec / test) .......... 5
E. Image and Shadow (64 Mb, 1 sec / test) .......... 6
F. Magic Square (64 Mb, 3 sec/test) .......... 8
G. GLONASS (64 Mb, 1 sec/test) .......... 10
H. Space competition (64 Mb, 3 sec/test) .......... 12
I. Color game (64 Mb, 2 sec / test) .......... 14
J. Serial Numbers (64 Mb, 4 sec/test) .......... 16
K. Processor's game (64 Mb, 1 sec/test) .......... 18

| | |
|---|---|
| **Input file name** | INPUT.TXT |
| **Output file name:** | OUTPUT.TXT |

## *A. Papa Carlo (64 Mb, 1 sec / test)*

Papa Carlo was making Buratino all his life. He made dozens, hundreds or maybe thousands of beautiful and well-dressed boys with long noses. Papa Carlo had a bunch of wooden sticks in his barn. He kept them for future noses. Last Friday Papa Carlo received an urgent order to produce batch of Buratino with the same noses. He decided to use all his wooden sticks for the noses.

For this, Papa Carlo measured all sticks and found that their lengths are natural integers. Then, the Master began to choose arbitrary two sticks of different length and saw off the longer one by the length of the smaller one. As a result three sticks were turned out. All of these sticks were sent back to the pile. So Papa Carlo had been working until all the sticks in the pile became the same length.

Help Papa Carlo and count the number of noses!

Write a program that using the number of sticks $\boldsymbol{n}$ and their lengths $\boldsymbol{l_i}$ ($\boldsymbol{i} = 1, ..., \boldsymbol{n}$) will determine the resulting number of noses.

**Example.** Let Papa Carlo has two sticks with the lengths of 4 cm and 6 cm. After the first sawing there will be three sticks: one 2cm stick and two 4cm sticks. After the second sawing there will be four sticks with the lengths of 2, 2, 2 and 4cm. Finally we will get five pieces of equal length 2 cm.

### Limitations

$1 \le \boldsymbol{n} \le 10000$; $1 \le \boldsymbol{l_i} \le 10000$; $1 \le \boldsymbol{i} \le \boldsymbol{n}$.

### Input

The first line contains an integer $\boldsymbol{n}$ – number of nose sticks. In the following $\boldsymbol{n}$ lines $\boldsymbol{n}$ integers are given which are the lengths of sticks, one number per line.

### Output

The output file should contain single integer $\boldsymbol{k}$ – number of sticks in the pile by the end of Papa Carlo's work.

### Example

| Input.txt | Output.txt |
|---|---|
| 3<br>2<br>7<br>5 | 14 |

| Input.txt | Output.txt |
|---|---|
| 2<br>6<br>4 | 5 |

## *B. ~~Not~~ so trivial problem (64 Mb, 1 sec / test)*

In XML some characters are strictly predefined and have special meanings. For instance, character «<» is used for the beginning of start-tags and end-tags. XML provides facilities for including text with special characters. Text is transformed as follows: each special character is replaced by the predefined entity «`&entity_name;`». There are five predefined entities: `&lt;` represents «<» (less), `&gt;` represents «>» (greater), `&amp;` represents «&» (ampersand), `&apos;` represents «′» (single quote), `&quot;` represents «″»(double quote). Let us call described transformation as function `XMLEncode(S)`, where ***S*** is an input string. `XMLEncode()` could be applied to input several times.

For example, let ***S*** be equals to «`2<3<4`». Then `XMLEncode(S)` will be «`2&lt;3&lt;4`» and `XMLEncode(XMLEncode(S))` will be «`2&amp;lt;3&amp;lt;4`».

Write a program that, by the resulting string ***R***, will determine the maximum number of XMLEncode-function calls.

### Limitations

1 ≤ length(***R***) ≤ 100000.

### Input

The single line of input file contains the string ***R***. The string can contain characters with ASCII codes from 32 to 127.

### Output

The output file should contain a single integer – number of XMLEncode-function calls, or -1 if the string could be encoded infinitely.

| Input.txt | Output.txt |
|---|---|
| `Everybody knows that 2 < 3.` | `0` |

| Input.txt | Output.txt |
|---|---|
| `Everybody knows that 2 &lt; 3.` | `1` |

## *C. Sort by sum of digits (64 Mb, 1 sec / test)*

The sum of a number’s digits associated with some of its properties. Particularly, by the sum of digits it is easy determine divisibility by 3 and 9. Computer scientists of Rybinsk have decided to do research of natural numbers and their sums of digits.

It is well known, that it’s easier to work with the sorted data than with unsorted. Therefore, first of all researchers decided to sort natural numbers by the sum of their digits. If some numbers have equal sums of digits, then these numbers are written in ascending order.

When scientists sorted the numbers from 1 to 20, they have got a series: 1, 10, 2, 11, 20, 3, 12, 4, 13, 5, 14, 6, 15, 7, 16, 8, 17, 9, 18, 19.

For the small quantity of numbers sorting were executed quickly. When they moved research in a wide range some difficulties were appeared to overcome. One of it was to find out the place of number K in the sorted series of ***N*** natural numbers. Your task is to solve the problem.

Write a program to calculate the position of K in the sorted series of the first ***N*** natural numbers.

### Limitations

$1 \le \boldsymbol{K} \le \boldsymbol{N} \le 10^{12}$.

### Input

The single line of the input file contains two natural numbers separated by space: ***N*** and ***K*** in their decimal notation.

### Output

One natural number – position of ***K***.

### Sample

| Input.txt | Output.txt |
|---|---|
| `20 13` | `9` |

## *D. Roman numerals (64 Mb, 1 sec / test)*

Roman numbers are based on seven symbols «I», «V», «X», «L», «C», «D» and «M». The «costs» of those symbols are 1, 5, 10, 50, 100, 500 and 1000, respectively. To calculate decimal value of a Roman number, we will use the following algorithm:

1. Find the most significant (the most «expensive») symbol. If there are several such symbols take **the leftmost one**. Let the found symbol is placed in position ***i***.
2. Denote as ***Middle*** the «cost» of symbol placed in position ***i***.
3. Calculate the value of Roman number formed by the symbols, standing to the right of ***i***. Denote this value as ***Right***.
4. Calculate the value of Roman number formed by the numerals, standing to the left of ***i***. Denote this value as ***Left***.
5. Let ***Middle*** + ***Right*** – ***Left*** be the value of a Roman number.

Obviously, if we use this algorithm, the same number can be written in many ways. For example, the number 19 can be written as «IXX», «XIX», «XVIV», «XVIIII» and so on.

Given decimal number ***n*** and Roman number ***S***, you are to find such digits transposition of ***S***, that represents ***n*** in Roman notation.

### Limitations

$-50000 \le n \le 50000$, $1 \le \text{length}(S) \le 50$

### Input

The first line contains an integer ***n*** in decimal notation. The second line contains a number ***S*** in Roman notation

### Output

The first line of output file should contain the Roman number or word «NO» (without the quotes) if there is no solution.

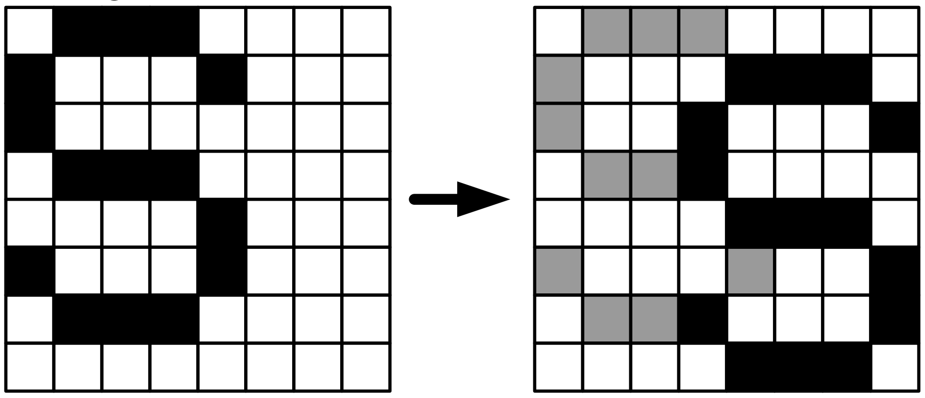
### Sample

| Input.txt | Output.txt |
|---|---|
| 16<br>XXVI | XVXI |

| Input.txt | Output.txt |
|---|---|
| 15<br>XXVI | NO |

## *E. Image and Shadow (64 Mb, 1 sec / test)*

In order to make machine recognition of characters more difficult in the «**CAPTCHA**» (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) system different transformation are used. One of these transformations is the «fall of the shadow» – the symbol is shifted and imposed on itself. For example, the picture below shows the process of falling the shadow with the letter «S» for an 8x8 image.



Let us have an 16x16 image. We take as the origin the upper left corner and we measure the horizontal ***x*** coordinate axis from the left to right and the vertical ***y*** coordinate axis top-down. We denote by (***x***, ***y***) vector, on which the image was shifted, and call it the shift vector. In the picture above, the shift vector is (3, 1) – image was shifted by 3 points to the right and one to the down. The original picture without imposed shadow is called the image and the modified image – image with shadow. We also agree that in the process of applying shadows, the shift vector is chosen in such way that shaded (black) point of the image does not fall outside the square 16x16.

Your task is to write a program that performs the inverse transformation: according to the image with the shadow find the image and the shift vector with the minimal numbers of shaded (black) points in the image. If there are several such images, the answer would be any of them.

### Limitations

Shift vector is limited so that black point of the image does not extend beyond the square 16x16.

$-15 \le \boldsymbol{x}, \boldsymbol{y} \le 15$

### Input

The input file contains an image with a shadow, and consists of 16 rows, each of which contains exactly 16 characters. Blank (white) point is denoted by «`.`» (ASCII = 2Eh), shaded point is denoted by «#» (ASCII = 23h).

## Output

The first line of the output file should contain two integers ***x*** and ***y*** (shift vector), separated by the space. Then 16 lines of 16 characters each should follow – the image encoded in the same way as in the input file.

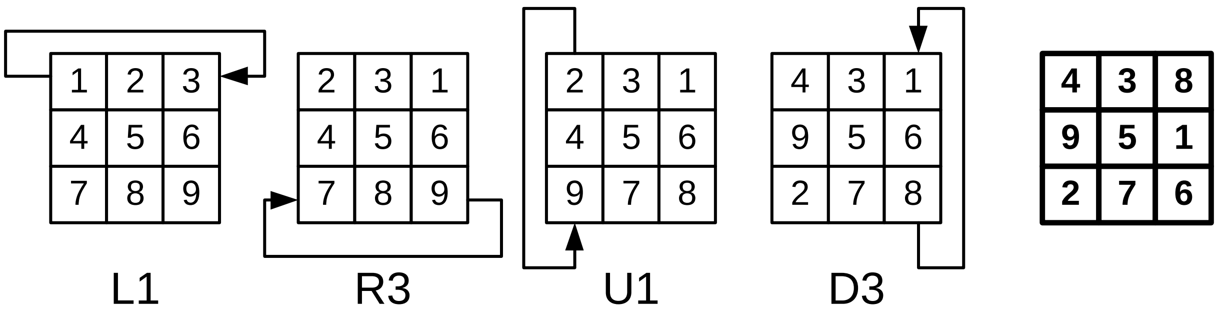
## Sample

| Input.txt | Output.txt |
|---|---|
| `.###............` | `3 1` |
| `#...###.........` | `.###............` |
| `#..#...#........` | `#...#...........` |
| `.###............` | `#...............` |
| `....###.........` | `.###............` |
| `#...#..#........` | `....#...........` |
| `.###...#........` | `#...#...........` |
| `....###.........` | `.###............` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
| `................` | `................` |
|  | `................` |

## *F. Magic Square (64 Mb, 3 sec/test)*

Numbers from 1 to 9 are written in 3x3 square at arbitrary positions. You can make cyclic shift by one element of some row (to the left or to the right) or column (to the up or to the down). The task is to implement a program, that could find a minimal number of shifts required to transform given square to the magic square[1].

There is example of such transformation on the picture:



### Limitations

Note that any square could be transformed to the magic square with some number of cyclic shifts.

### Input

There are three lines in the input file. Each line represents row of the square and consists of three numbers separated by spaces.

### Output

The first line of output file should contain integer ***n*** – the minimal number of shift operations required to transform given square to the magic one. First line is followed by ***n*** lines, each of them describe single shift. Shifts are encoded in such a way:

- D*x* – shift down the column *x*;
- U*x* – shift up the column *x*;
- L*x* – shift left the row *x*;
- R*x* – shift right the row *x*.

Row (column) number *x* is written after symbols «D», «U», «L» and «R» without space, $1 \le x \le 3$. Rows of the square are numbered from up to down, columns – from left to right.

### Sample

| Input.txt | Output.txt |
|---|---|
| `1 2 3` | `4` |

[1] A "magic square" is a rectangular array of numbers, where each column, row, and both diagonals have the same sum (for our case sum is equal 15).

| | |
|---|---|
| 4 5 6<br>7 8 9 | L1<br>R3<br>U1<br>D3 |

## *G. GLONASS (64 Mb, 1 sec/test)*

Nowadays there are two global positioning systems in the world, that allow us to determine exact position and some other parameters of object that located on the ground. These are **GPS** (**G**lobal **P**ositioning **S**ystem) and **GLONASS** (**Glo**bal **S**atellite **Na**vigation **S**ystem). Also there is the “Beidou” (北斗導航系統) system, that is being developed today.

These systems are based on the main principle – the position is determined by measurements of the distance to the object from the satellites, which coordinates are known. Distances are calculated using the time delay between the moment, when signal was sent from satellite and the moment, when signal was received by GPS or GLONASS receiver. To determine three-dimensional coordinates of the object one should know the distance from the object to at least three satellites. Moreover satellites should be in direct "visibility" of the object, and placed "around" the object.

The latter concept is specified by experts as follows: object must be strictly inside the tetrahedron, the three vertices of which coincide with the three given satellite, and the fourth is at the center of the Earth.

Since the satellites are constantly changing their position because of the orbital motion, and the object could be located in any geographical point, the total number of satellites in the system must be large enough so that one always be able to find three that meet specified conditions.

Software and hardware of global positioning systems is rather complex. Fortunately, you have to implement only a small part of it.

Assume that the sea level is at the distance of 6000 km from the center of the Earth. It is guaranteed that the object is strictly inside the tetrahedron with vertices at the center of the earth and satellites.

Write a program calculating the height of the object above sea level, given the coordinates of three satellites and their distances from the object.

### Limitations

All coordinates and distances are integers, given in meters, and can range from $-10^8$ to $+10^8$.

### Input

Each of the first three lines consists of three numbers – the Cartesian coordinates of the first, second and third satellites, respectively. Center of the Earth is at the point (0, 0, 0).

There are three numbers separated by spaces in the fourth line – distances from the object to the first, second and third satellite, respectively.

### Output

The output file consists of one number with an accuracy of 0.5 meters.

**Sample**

| **Input.txt** | **Output.txt** |
|---|---|
| `6000031  40  0`<br>`6000031  0  40`<br>`6000031  -24  -32`<br>`50  50  50` | `1` |

***Note.*** The coordinate of the object in sample: x=6000001, y=0, z=0

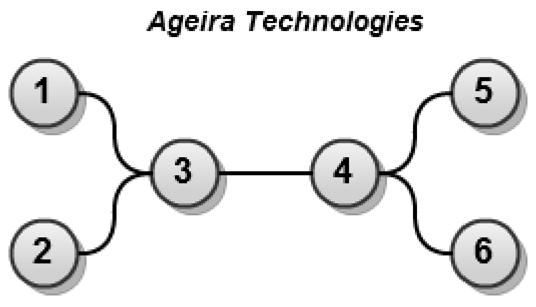
## *H. Space competition (64 Mb, 3 sec/test)*

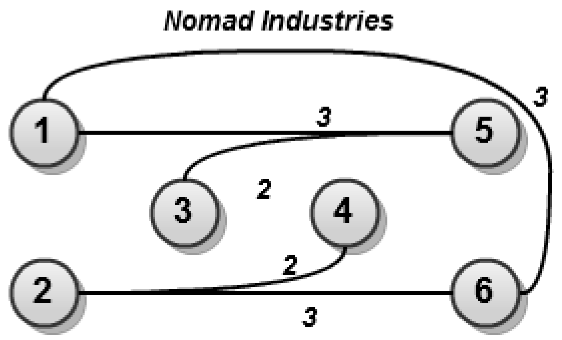
***N*** planets are situated in the sector E4 of the Omicron Theta area. And it just so happened that the Ageira Technologies company has completely monopolized space transportation by building a network of Trade Lines.

Each Trade Line is a two-way portal between a pair of planets. Ageira Technologies, taking advantage of the economic situation, built a minimal number of Trade Lines so that one could get from each planet ***i*** to any other planet ***j***, using some of Trade Lines. There are ***n***-1 Trade Lines built by Ageira.

Many residents of distant planets were unhappy with such situation, because to get from one planet to another, they have to use a lot of Trade Lines. And the longer the journey, the greater the danger to meet space pirates!

Nomad Industries company, the rival of Ageira company, wants to take advantage of the situation and build in the sector theirs own network of Hyper Gates to capture part of the market of space transportation. Qualified specialists of Nomad Industries has found out that profit from building the Gate between planets ***i*** and ***j*** will be equal to the length of the path that must be overcome by using only Ageira‘s Trade Lines. By length of the path between planet ***i*** and planet ***j*** we mean the minimal number of Trade Lines used to get from planet ***i*** to planet ***j***.





Nomad Industries has decided to act on the principle similar to Ageira's policy: to build a minimal amount of the Gates in such a way that one could get from each planet ***i*** to any other planet ***j***, not using the Trade Lines at all.

Write program, that will find such Gate configuration, which has a maximal resulting profit of all gates built. (Note, that company should build exactly ***N***-1 Gates)

## Limitations

$2 \le N \le 2000$

## Input

The first line of input file consists of an integer ***N*** – the number of planets in the sector and the number of Trade Lines, respectively. First line is followed by ***N***-1 lines, each contains two numbers separated by a space – planets' identifiers, connected by Ageira Technologies' Trade Lines. Identifiers of planets are the unique integers from 1 to ***N***.

## Output

The first line of output file should contain single integer – the maximal profit of Nomad Industries company.

## Sample

| Input.txt | Output.txt |
|---|---|
| 6<br>1 3<br>2 3<br>3 4<br>4 5<br>4 6 | 13 |

## *I. Color game (64 Mb, 2 sec / test)*

Vasya plays an intricate game. Several circles with colored points (each circle have the same number of points) lies before him. Note that circles themselves are located around the circle. All circles are numbered from 1 to ***N*** and points on these circles – from 1 to ***M***. Each point is colored in one of 30 colors.

Initially, Vasya puts a token on the first point of the first circle. Then he moves the token ***T*** times as follows:

1. Let the token be in the circle ***C*** at the point with the number ***P***.

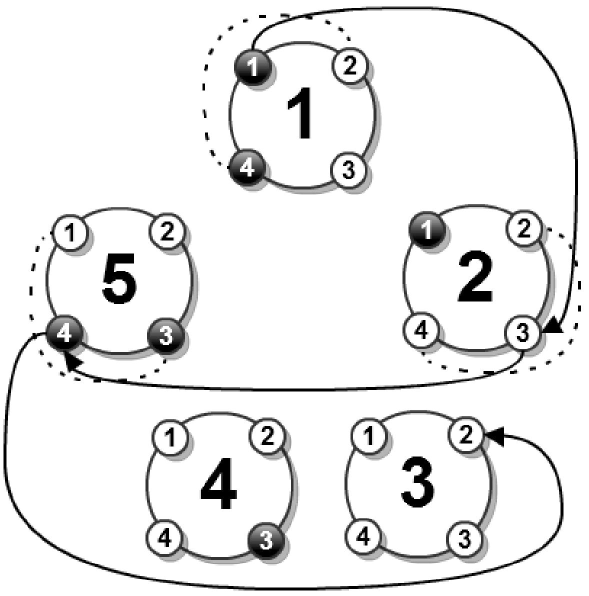
2. Let ***S*** be the number of points of different colors in the ***K***-neighborhood of ***P***-point on the circle ***C***, i.e. on the segment [***P***-***K***, ***P***+***K***]. Note that the segment is located on the circle, so point ***M*** is followed by point 1. For example, if ***M*** = 6, ***P*** = 5, ***K*** = 2, then the segment is [***P***-2, **P**+2], which breaks down to [3, 6] and [1, 1].

3. Move the token to the circle (***N***-***C***+1)+***S*** at the point ***P***+***S***. Note that circles themselves are located around the circle and circle ***N*** is followed by circle 1.

Vasya wants to find out at which circle and point token will be after all the trick moves.

Consider an example. Let ***N*** = 5, ***M*** = 4, ***K*** = 1. The example uses only two colors: black and white. The dotted lines on the picture denote the neighborhood of the points at which the token was.

Initially, ***C*** = 1, ***P*** = 1. In the 1-neighborhood there are both colors, i.e. ***S*** = 2. Hence ***C'***= (5-1+1)+2 = 2, ***P'*** = 1+2 = 3. In this new position in the 1-neighborhood there is only the white color, so ***S*** = 1. ***C''***= (5-2+1)+1=5, ***P''***=3+1=4. Finally, in the next-to-last position in the 1-neighborhood we have both colors, ***S*** = 2, ***C'''***=(5-5+1) +2 = 3, ***P'''***=4+2 = 2.



For a given set of circles and the number ***T*** you are to find where the token will be after ***T*** moves.

Note, that given circles have one feature – for i = 1,2, ..., ***N***-1 circle ***i*** differs from the circle ***i***+1 in at most one point.

## Limitations

$1 \le N \le 50000$, $1 \le T, M \le 100000$, $0 \le K < M/2$.

## Input

The first line contains four numbers ***N***, ***M***, ***T***, ***K***. The second line contains ***M*** integers – the colors of points on the first circle. Each color is an integer ranging from 0 to 29.

The next ***N***-1 rows consist of difference between successive circles: the row ***i*** describes the difference between the circle ***i*** and circle ***i***+1 – a pair of numbers $P_i$, $Color_{i+1}$ – the point at which the circles different and color of the point for the circle ***i***+1.

## Output

The output file should contain a pair of numbers – the number of circle and the number of point – position of the token after ***T*** moves.

## Sample

| Input.txt | Output.txt |
|---|---|
| 5 4 3 1<br>0 1 1 0<br>4 1<br>1 1<br>3 0<br>4 0 | 3 2 |

## *J. Serial Numbers (64 Mb, 4 sec/test)*

To check a validity of the software licenses company M* uses serial numbers. Each serial number is a string consisting of exactly ***n*** hexadecimal digits. To recognize the correct serial number, firm M* uses a state machine. That state machine has ***k*** states $S_1$, $S_2$, ..., $S_k$ and ***r*** transitions between them. Transitions are marked by letters, and define the rules of transition from one state to another. The state machine constructed so that all the transitions leading from state $S_i$ to other states, necessarily are marked by different letters (that kind of state machine is called deterministic).

Machine starts with the state $S_1$ and sequentially process ***n*** letters of the serial number. Lets assume, that at the moment state machine is in state $S_i$ and process ***j***-th letter of serial number, then:

1. If there is a transition from the state $S_i$ that is marked with the processed letter and leading to the state $S_k$, then go to the state $S_k$ and begin processing (***j*** +1)- th letter.

2. If there is no transition from the state $S_i$ labeled with the ***j***-th letter, then stop processing and assume that the serial number is invalid.

The serial number is valid, if all its letters were successfully processed and the machine returned to the state $S_1$.

Your task is to determine the number of different serial numbers that the given state machine could recognize as valid.

### Limitations

$2 \le k, n \le 50$; $1 \le r \le 800$.

$1 \le S_i, F_i \le k$, where $i = 1, \dots r$.

$\forall\ (S_i, F_i, L_i), (S_j, F_j, L_j)$: $S_i = S_j \Rightarrow L_i \ne L_j$, where $i, j = 1, \dots, r$.

### Input

The first line contains three integers ***k***, ***n***, ***r***, separated by spaces. First line is followed by ***r*** rows, each consisting of three numbers $S_i$, $F_i$, $L_i$. The $S_i$ integer is the start state of ***i***-th transition, and the integer $F_i$ – the end state of the transition. The hexadecimal digit $L_i$ is the label of the ***i***-th transition.

### Output

The output file should contain a single integer number – the serial numbers' count, which state machine recognize as valid.

## Sample

| Input.txt | Output.txt |
|---|---|
| `3 4 6`<br>`1 2 0`<br>`1 3 1`<br>`2 1 0`<br>`2 3 3`<br>`3 1 4`<br>`3 2 2` | `6` |

**Note.** Sample state machine recognize following serial numbers: «0000», «0014», «0320», «1234», «1400» and «1414».

## *K. Processor's game (64 Mb, 1 sec/test)*

Vasya Pupkin' coursework on electronics is to create a simple microprocessor. According to the statement this microprocessor has only three registers: the accumulator, the instruction pointer and stack pointer. The accumulator has 60 bits, the instruction and stack pointers has 7 bits. The accumulator can store positive and negative numbers (in binary complement code[2]).

The processor can execute the following commands (ACC – the accumulator, IP – instruction pointer, SP – stack pointer, [X] – element of the stack with the address X):

| Mnemonic | Meaning | Pseudocode |
|---|---|---|
| ADD X | Binary addition the accumulator value with a constant. When overflow extra bits are discarded | ACC $\leftarrow$ (X + ACC) mod $2^{60}$<br>IP $\leftarrow$ IP + 1 |
| SUB X | Binary substraction of the accumulator value **from** a constant | ACC $\leftarrow$ (X – ACC) mod $2^{60}$<br>IP $\leftarrow$ IP + 1 |
| NEG | Changing the sign of the accumulator register value (binary complementation) | ACC $\leftarrow$ (–ACC) mod $2^{60}$<br>IP $\leftarrow$ IP + 1 |
| NOT | Bitwise negation of the accumulator value | ACC $\leftarrow$ not (ACC)<br>IP $\leftarrow$ IP + 1 |
| CALL R | Calling the subroutine that starts at the address R | SP $\leftarrow$ SP + 1<br>[SP] $\leftarrow$ IP + 1<br>IP $\leftarrow$ R |
| RET | Returning from a subroutine or stop | if SP = 0 then<br>  Stop;<br>IP $\leftarrow$ [SP]<br>SP $\leftarrow$ SP – 1 |

The program is located in cells with addresses from 0 to 127. Each memory cell can contain exactly one command (including the constant value, if it exists). Before beginning execution, the accumulator is set to the value to be processed, and registers IP and SP are zeroed. The result of calculation is the value of the accumulator after the program stops.

Given microprocessor program, write a program that calculates its result for each provided accumulator value.

### Limitations

$1 \le \boldsymbol{n} \le 128$; $1 \le \boldsymbol{m} \le 10^3$, $-2^{59} \le \boldsymbol{a_i} < 2^{59}$.

[2] Binary complement code of negative number is equal to difference between $2^{60}$ and absolute value of that negative number. For example, -5 in decimal complement code could be written as 1152921504606846971, and in binary complement code as 111111111111111111111111111111111111111111111111111111111011.

All mnemonics are written with capital Latin letters, and a constant X is specified as a decimal number (possibly signed) ranged from $-2^{59} \le \boldsymbol{X} < 2^{59}$; for the CALL command $0 \le \boldsymbol{X} \le 127$.

The given program has a finite run time, and never overflows the stack.

## Input

The first line contains two integers ***n*** and ***m***, separated by spaces, where ***n*** – number of commands in the program, ***m*** – number of values of accumulator for which you should make calculations. The subsequent ***n*** lines contain the commands in the following format:

"Address of memory cells," "space" "command's mnemonic" ["space", "argument."]

Command are listed sequentially from cell with address 0 to cell with address ***n***-1.

The program is followed by ***m*** integers $\boldsymbol{a_i}$ – initial values of the accumulator, which should be processed by microprocessor. The numbers are separated by spaces.

## Output

The first line of the output file should contain integer ***m*** – the number of calculated values. Following ***m*** lines should contain the results of calculations for input values $\boldsymbol{a_i}$, placed in the same order as $\boldsymbol{a_i}$,, one per line.

## Sample

| Input.txt | Output.txt |
|---|---|
| 6 2 | 2 |
| 0 ADD 9 | 13 |
| 1 CALL 4 | 22 |
| 2 SUB 3 | |
| 3 RET | |
| 4 NEG | |
| 5 RET | |
| 1 10 | |
| **Input.txt** | **Output.txt** |
| 5 2 | 2 |
| 0 CALL 1 | 19 |
| 1 CALL 3 | -82 |
| 2 ADD 5 | |
| 3 ADD 2 | |
| 4 RET | |
| 1 -100 | |

**Rybinsk State Aviation Technical University**

***Task authors:***

| | |
|---|---|
| ***© Sergey G. Volchenkov, 2011*** | **(volchenkov@yandex.ru)** |
| ***© Vladimir N. Pinaev, 2011*** | **(vpinaev@mail.ru)** |
| ***© Michael Y. Kopachev, 2011*** | **(rybkmu@mail.ru)** |
| ***© Alexander Kiselyov, 2011*** | **(diver.ru@gmail.com)** |
| ***© Alexander Kouprin, 2011*** | **(KouprinAl@yandex.ru)** |