Quarterfinal

Central region of Russia

Rybinsk, October 14-15-2009

Problems

A. Numeric sequence (16 Mb, 1 sec) 2
B. Triangle (16 Mb, 1 sec) 3
C. Tower of Hanoi (16 Mb, 1 sec) 4
D. Lucky tickets (16 Mb, 1 sec) 6
E. Optimization (16 Mb, 5 sec) 8
F. Wordplay (16 Mb, 1 sec) 11
G. Traffic (24 Mb, 3 sec) 12
H. Reform (16 Mb, 1 sec) 14
I. Brackets (8 Mb, 1 sec) 15
J. Sequence strikes back! (16 Mb, 1 sec) 16

Input file:	**INPUT.TXT**
Output file:	**OUTPUT.TXT**

A. Numeric sequence (16 Mb, 1 sec)

An infinite numeric integer sequence ***F*** is defined by the following relations:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_i = F_{i-1} + F_{i-2} \end{cases}$$

Write a program that calculates the value of ***n***-th element in the sequence ***F***.

Input

The first line of the input file contains one integer ***n*** – the index of the element in the sequence.

Output

The output file should contain one integer – the value F_n.

Limitations

–40 <= ***n*** <= +40

Sample Input 1	**Sample Output 1**
1	1
Sample Input 2	**Sample Output 2**
10	55

B. Triangle (16 Mb, 1 sec)

During his summer holidays, hacker Kirill helped his parents in the country. He was to take care of a potato field of triangular shape. One day, when he was picking potato beetles and thinking about the use of geometry, he invented the following problem.

Three points on the plane are defined by their coordinates: $A(X_a; Y_a)$, $B(X_b; Y_b)$ and $C(X_c; Y_c)$. Find point O inside triangle ABC such that the proportion of areas of triangles AOB, BOC and COA would be $P : Q : R$.

Help Kirill and write a program to determine the coordinates of point O.

Limitations

$-10000 \le X_a, Y_a, X_b, Y_b, X_c, Y_c \le 10000$

$1 \le P, Q, R \le 10000$.

Numbers $X_a, Y_a, X_b, Y_b, X_c, Y_c, P, Q, R$ are integer.

Triangle ABC is nondegenerate.

Input

The first three lines of the input file contain the coordinates of points A, B, C: $X_a, Y_a, X_b, Y_b, X_c, Y_c$.

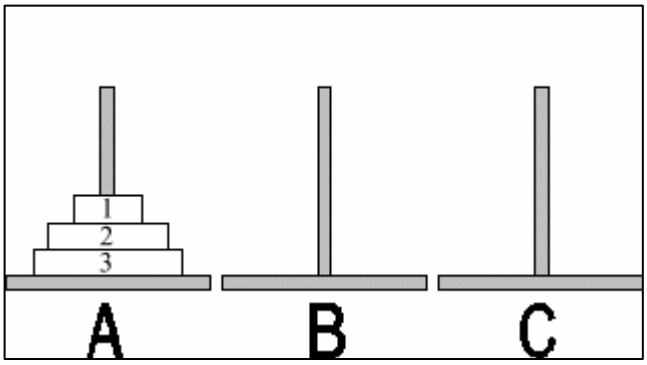
The fourth line lists numbers P, Q and R.

Output

Write the coordinates of point O with precision 10^{-6} to the output file.

Sample Input	Sample Output
0 0 6 8 6 0 7 8 9	4 3

C. Tower of Hanoi (16 Mb, 1 sec)



Let us briefly remind the conditions of this well-known puzzle.

There are 3 rods marked **A**, **B**, **C**. Initially, ***N*** disks of different diameter are stacked in order of size on rod **A,** with the smallest disk at the top. The second and the third rods are empty yet.

The objective is to move all the disks from rod **A** to rod **B**, using rod **C** as auxiliary.

On each move you can take the top disk from any rod and then put it either on an empty rod or on a rod where the top disk is larger than taken.

Virtually all books on programming describe a recursive solution of this puzzle. The following example demonstrates a procedure written in Pascal.

```
Procedure Hanoi (X, Y, Z: char; N: integer);
Begin
 If N>0 then
    Begin
       Hanoi (X, Z, Y, N-1);
       Writeln('Disk ', N, ' from ', X, ' to ', Y);
       Hanoi (Z, Y, X, N-1)
    End
End;
```

Note that when the puzzle is being solved the disks may be moved in the following directions[1]: AB, AC, BA, BC, CA, CB. The example demonstrates the number of moves in each possible direction.

[1] A direction is defined by initial and final rod letters.

Example. N=3

Move No.	Disk	Rod		Direction
		From	To	
1	1	A	B	AB
2	2	A	C	AC
3	1	B	C	BC
4	3	A	B	AB
5	1	C	A	CA
6	2	C	B	CB
7	1	A	B	AB

Direction, S (for N=3)	AB	AC	BA	BC	CA	CB
Number of moves	3	1	0	1	1	1

Write a program to calculate the number of moves for given number of disks N and direction S.

Limitations

1 <= N <= 40

Input

The first line of the input file contains a single integer N, the number of disks, and the second line contains two capital letters S, the direction in question.

Output

The output file must contain a single integer, the number of moves in the given direction S.

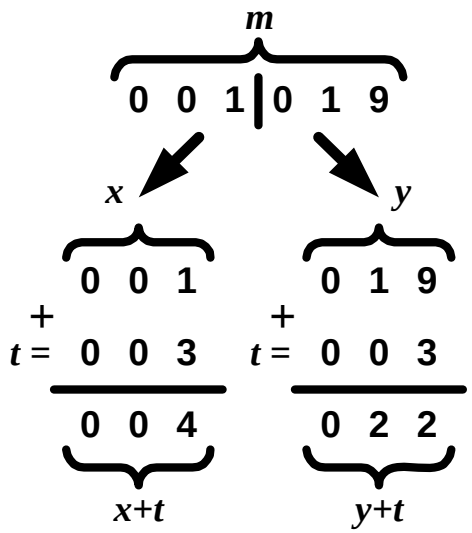
Sample Input 1	**Sample Output 1**
3 AB	3
Sample Input 2	**Sample Output 2**
3 CA	1

D. Lucky tickets (16 Mb, 1 sec)

The problems concerning lucky tickets are considered classic. Let us remind that a ticket is lucky if its number ***S*** satisfies certain conditions. Let the number of the ticket have ***m***=2****n*** digits. We consider a ticket to be lucky if the sum of the first ***n*** digits is equal to the sum of the last ***n*** digits. *If the number of a ticket does not meet this condition we call such a ticket unlucky.*

To turn an unlucky ticket into a lucky one, the following process is proposed:

1. denote the number formed by the first ***n*** digits as ***x***, and the number formed by the rest ***n*** digits as ***y***;
2. add a positive integer ***t*** to both ***x*** and ***y***; ***t*** is chosen so that the number of digits in ***(x+t)*** and ***(y+t)*** is still ***n***, and the sums of digits of the new numbers are equal;



Write a program that will calculate positive number ***t*** for given ticket number ***S***. Number ***t*** must turn an unlucky ticket into a lucky one according to the process above. If several such numbers exist then any of them will be the answer. If there are no such numbers then output -1 as the answer.

Limitations

$2 \le 2*n \le 100$;
$0 \le S \le 10^{2*n} - 1$

Input

The input file consists of a single line with number ***S*** (unlucky ticket number) having ***m***=2****n*** decimal digits.

Output

The output file contains a single integer ***t***. If no solution exists then the output file contains –1.

Sample Input 1	**Sample Output 1**
001019	3
Sample Input 2	**Sample Output 2**
98	-1
Sample Input 3	**Sample Output 3**
654204	46

E. Optimization (16 Mb, 5 sec)

Kirill is a hacker and he learns programming at the academy. He is fond of trying various pieces of destructive software against the LAN of his *alma mater*.
In order to make the talented one busy and give some rest to unfortunate system administrators, Kirill's teacher assigned him a challenging course project.
The goal of the project is to develop an algorithm to optimize programs written in assembly language of a simple CPU.
A program consists of a sequence of lines numbered 1 to ***n***. Program execution starts with the line number 1, and stops on line ***n***, which is guaranteed to contain the **RET** command. Program lines are executed sequentially, one after another. The order of execution may be altered only by special jump commands.
The CPU has four eight-bit registers labeled **A**, **B**, **C**, and **D**. Each register can store an integer from the range [0, 255]. Aside from the registers the CPU has two flags: zero flag **ZF** and carry flag **CF**. Possible flag values are 0 or 1. There is also a special command register **R**. This register stores the number of the line being executed.
The following table lists all commands supported by this CPU.

Mnemonic	**Command description**	**CPU actions**
MOV reg1, reg2	Move data	reg1 ← reg2
LD reg1, #const	Load a constant into a register	reg1 ← #const
ADD reg1, reg2	Add	CF ← (reg1 + reg2) div 256; reg1 ← (reg1 + reg2) mod 256; ZF ← [reg1 = 0]
ADC reg1, reg2	Add with carry	CF ← (reg1 + reg2 + CF) div 256; reg1 ← (reg1 + reg2 + CF) mod 256; ZF ← [reg1 = 0]
NEG reg	256 complement	reg ← 256 – reg; ZF ← [reg = 0]; CF ← 1
AND reg1, reg2 OR reg1, reg2 XOR reg1, reg2	Bitwise boolean operations "and", "or" and "exclusive or"	reg1 ← reg1 & reg2; (AND) reg1 ← reg1 | reg2; (OR) reg1 ← reg1 xor reg2; (XOR) ZF ← [reg1 = 0]; CF ← 0
INC reg	Increment register	reg ← reg +1; ZF ← [reg = 0]
DEC reg	Decrement register	reg ← reg – 1; ZF ← [reg = 0];
CLC	Clear carry flag	CF ← 0
STC	Set carry flag	CF ← 1

JUMP #xxx	Unconditional jump to line #xxx	R ← #xxx
JZ #xxx	Jump to line #xxx if zero flag (ZF) is set	if ZF = 1 then R ← #xxx
JNZ #xxx	Jump to line #xxx if zero flag (ZF) is not set	if ZF = 0 then R ← #xxx
JC #xxx	Jump to line #xxx if carry flag (CF) is set	if CF = 1 then R ← #xxx
JNC #xxx	Jump to line #xxx if carry flag (CF) is not set	if CF = 0 then R ← #xxx
RET	Finish execution	Stop

Character "←" in the table stands for assignment, "div" for integer division, "mod" for remainder of division. Expression "[*reg* = 0]" evaluates to 1 if register *reg* contains zero, and 1 otherwise.

The following actions are allowed for program optimization:

- change the order of lines, except for the last line, which must always contain the **RET** command;
- change type, condition and target line for jump commands;
- remove and/or add jumps of any kind if it does not influence the result of execution;
- remove lines that are never executed.

It is forbidden to add or remove other commands. The changes must not influence the program logic.

Write a program that will optimize an assembly language program by removing the largest possible number of jumps (JUMP, JZ, JNZ, JC, JNC) from it.

Limitations

Number of lines in the assembly language program, n: $1 \le n \le 100$.

It is guaranteed that no line of the input data is executed more than once.

Input

The input file consists of one or more lines. The file is ended by a line containing the **RET** command. Each line contains number ***i*** and a command. Number ***i*** is an integer without leading zeroes, separated from the command by exactly one space character. A command consists of a mnemonic (see table) and possible operands. The first operand (if applicable) is separated from the mnemonic by exactly one space character. The second operand (if applicable) is separated from the first one by a comma character, without spaces. Mnemonics and register names are written using only capital Latin letters. If an operand is an integer (line number for a jump, or constant for an **LD** command) then it is written as a decimal number.

The last line of the input file contains a single integer ***K***, the limit of optimization (the output program must contain at most ***K*** jumps).

Output

The output file must contain the text of the program after optimization, written according to the rules given for the input file. The output program must contain no more then ***K*** jumps.

Sample Input 1	**Sample Output 1**
1 MOV A,B 2 JUMP 5 3 LD B,10 4 JUMP 6 5 JUMP 3 6 RET 0	1 MOV A,B 2 LD B,10 3 RET
Sample Input 2	**Sample Output 2**
1 ADD A,B 2 JZ 4 3 JUMP 5 4 ADD B,C 5 RET 1	1 ADD A,B 2 JNZ 4 3 ADD B,C 4 RET
Sample Input 3	**Sample Output 3**
1 ADD A,A 2 LD B,1 3 AND A,B 4 JZ 6 5 JUMP 8 6 LD C,5 7 JUMP 9 8 LD C,6 9 RET 0	1 ADD A,A 2 LD B,1 3 AND A,B 4 LD C,5 5 RET

F. Wordplay (16 Mb, 1 sec)

Hacker Kirill is fond of playing with words. He considers any sequence of letters to be a word, even if the sequence makes no sense.

Once he amused himself by composing chains of words using characters **a** and **b** only. Each following word ***Q*** in the chain must be produced from the previous one ***P*** with the use of one of the following rules:

- append letter **a** to the left of ***P***, i.e. ***Q***=**a*P***;
- replace the sequence **bb**…**ba**… at the beginning of ***P*** with **aa**…**ab**… (the number of **b**'s in the first fragment is equal to the number of **a**'s in the second one, and it may be equal to zero);
- replace the whole word ***P***=**bb**…**b** with ***Q***=**aa**…**ab** (the number of **b**'s in ***P*** is equal to the number of **a**'s in ***Q***).

An example of word chain: **bb** → **aab** → **bab** → **abab**.

Write a program that determines if it is possible to build a chain to turn one given word (***S***) into another (***F***) and finds the number of transformations in the shortest chain.

Limitations

The input word ***S*** is 1 to 50 characters long and is guaranteed to end with **b**.

Input

The first line of the input file contains the initial word ***S***. The second line contains the target word ***F***. Words ***S*** and ***F*** differ.

Output

The input file should contain a single number, the number of transformations. If it is impossible to build the required chain then write -1 to the output file.

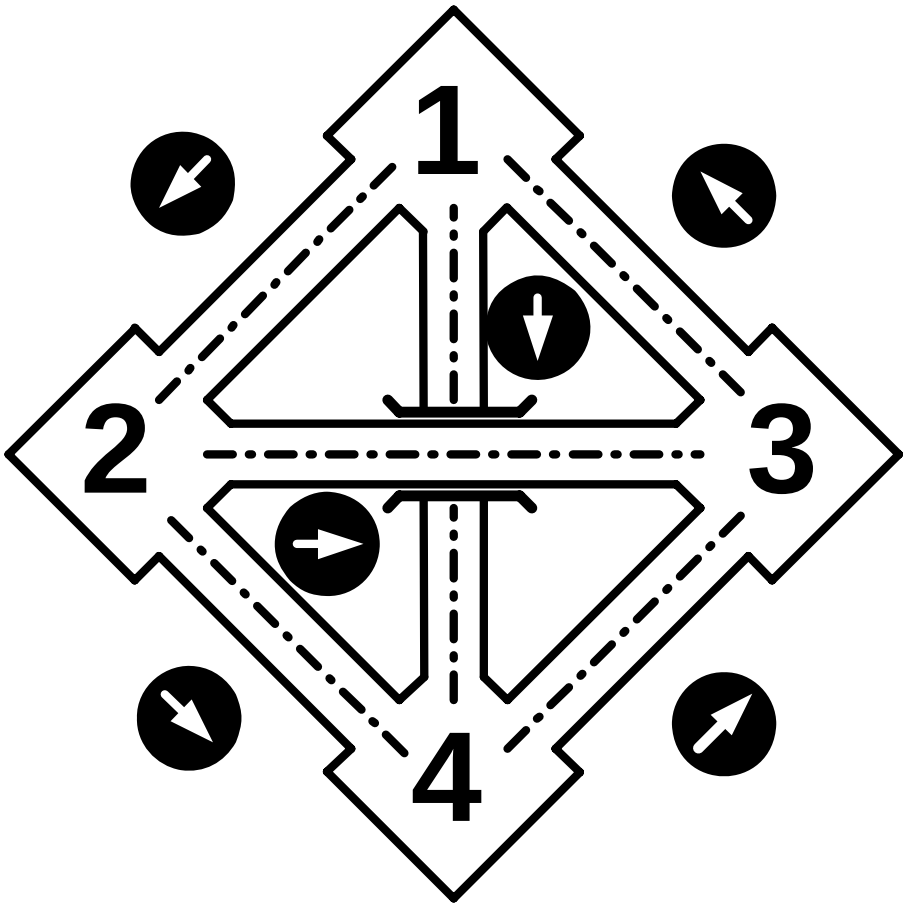
Sample Input	Sample Output
bb abab	3

G. Traffic (24 Mb, 3 sec)

Traffic accidents have become more frequent in the city of R-sk. Road repairs, building of overpasses and other measures did no good. Therefore, the mayor resorted to extreme measures of making all roads one-way. It is important to note that there had been no one-way roads in the city before.

There is a total of ***m*** roads and ***n*** road intersections in R-sk (these are numbered 1 to ***n***). Each road connects two different intersections. All intersections are achievable from each other. There is at most one road connecting any two intersections.

Help the mayor and write a program that will work out the layout of one-way roads for the given map of roads and intersections. The new one-way road layout must still allow for any two intersections to be mutually achievable.



Limitations

$2 \le n \le 1000$;

$1 \le m \le 1000000$.

Input

The first line of the input file contains two integers, ***n*** and ***m***, separated by spaces. Then ***m*** lines follow, containing a pair of integers each. Numbers in these pairs stand for intersections connected by roads.

Output

The output file should contain ***m*** lines, one for each road. Each line should contain a pair of numbers, the numbers of intersections connected by a one-way road (traffic direction is "first-to-second").
If it is impossible to work out the layout of one-way roads then **"No Solution"** (without quotation marks) should be written to the output file.

Sample Input 1	**Sample Output 1**
2 1 1 2	No Solution
Sample Input 2	**Sample Output 2**
4 6 1 2 1 3 1 4 2 3 2 4 3 4	1 2 1 4 2 3 2 4 3 1 4 3

H. Reform (16 Mb, 1 sec)

In Faraway Land there are a number of ministries. All the ministries are located along the same street and are numbered 1 to ***N***.
One day, the King decided to reform his government. He planned to dismiss each and every minister, and instantly appoint them to new ministerial positions. However, a condition was to be met: a minister can become the head of a new ministry if its distance from the previous ministry does not exceed ***K***. The court besought the King to set ***K***=0 but he noticed the catch because in this case the reform would have the only option: to appoint all ministers to the same positions. After some thought the King was close to not limiting ***K*** but the court astrologer (and mathematician) noted at once that in this case there would be ***N***! different variants, which is world known.
So, the King rejected his idea and decided to limit ***K***. Soon afterwards, he ordered to prepare the detailed plans of all possible appointments. Right by this time the neighboring kingdoms allowed Faraway Land to import a new supercomputer, and the hordes of programmers, which appeared during kingdom-wide computerization, were eager to do at King's behest.
We do not require you to fulfill the King's order in detail but try to find out the total number of distinct possible variants of the reform.
Write a program to calculate the total number of distinct reform variants for given ***N*** and ***K***.

Limitations

1 <= ***N*** <= 50
1 <= ***K*** <= 4

Input

Two integers ***N*** and ***K*** are given in the input file, on the first and second line, correspondingly.

Output

A single number should be written to the output file, the number of possible variants of the reform modulo 1,000,000 (i.e. the last 6 digits).

Sample Input 1	**Sample Output 1**
6 1	13
Sample Input 2	**Sample Output 2**
4 2	14
Sample Input 3	**Sample Output 3**
6 2	73

I. Brackets (8 Mb, 1 sec)

While attending boring lectures, hacker Kirill likes to play an entertaining game. He writes down a random sequence of ***n*** brackets (left and right, round and square). Kirill then tries to make this sequence balanced using a number of simple transformations:

- if a left round bracket is followed by a square one, left or right, they may be swapped: "([" → "[(" or "(]" → "](";
- if a right round bracket is preceded by a square one, left or right, they may be swapped: " [)" → ")[" or "])" → ")]";
- a pair of balanced sequential brackets may be replaced with a pair of brackets of other type "()" → "[]" or "[]" → "()".

For example, it is possible to get two balanced sequences from the sequence "([)]": "[()]" and "()[]".

Note that Kirill stops transformations as soon as he gets a balanced sequence.

Write a program to determine the number of distinct balanced bracket sequences that can be produced from the initial sequence with the use of transformations described above.

Limitations

$n \in \{2, 4, 6, 8, 10, 12\}$.

Input

The input file contains a single line, the initial sequence of brackets.

Output

The output file should contain an integer, the number of distinct balanced bracket sequences that can be produced from the initial one by given transformations.

Sample Input 1	**Sample Output 1**
`()`	`1`
Sample Input 2	**Sample Output 2**
`([)]`	`2`
Sample Input 3	**Sample Output 3**
`]]((`	`0`

J. Sequence strikes back! (16 Mb, 1 sec)

Hacker Kirill once saw two numeric sequences written on a fence: 8, 9, 10, 11 and 4, 3, 2, 1. While examining the numbers he noticed that each number from the first sequence is divisible by a corresponding number from the second one: 8 is divisible by 4, 9 is divisible by 3 and so on. Having found this interesting fact, Kirill went on to look for a sequence of nonnegative numbers ***X***, ***X***+1, ..., ***X***+***N***-1, such that ***X*** would be divisible by ***N***, ***X***+1 would be divisible by ***N***-1, ..., ***X***+***N***-1 would be divisible by 1, with the smallest possible ***X***.
Help Kirill and write a program to find ***X*** for given ***N***.

Limitations

1 <= ***N*** <= 40

Input

The first line of the input file contains a single integer ***N***.

Output

The output file should contain a single number, ***X***.

Sample Input	Sample Output
4	8

Rybinsk State Academy of Aviation Technology

Task authors:
© Sergey G. Volchenkov, 2009 (volchenkov@yandex.ru)
© Vladimir N. Pinaev, 2009 (vpinaev@mail.ru)
© Michael Y. Kopachev, 2009 (mkopachev@krista.ru)
© Victor Vinogradov, 2009 (fly3333@gmail.com)
© Dmitry A. Kormalev, 2009 (dkormalev@mail.ru)