

Problem #1: The Hardest Problem Ever

Introduction

Julius Caesar lived in a time of danger and intrigue. The hardest situation Caesar ever faced was keeping himself alive. In order for him to survive, he decided to create one of the first ciphers. This cipher was so incredibly sound, that no one could figure it out without knowing how it worked.

You are a sub captain of Caesar's army. It is your job to decipher the messages sent by Caesar and provide to your general. The code is simple. For each letter in a plaintext message, you shift it five places to the right to create the secure message (i.e., if the letter is 'A', the cipher text would be 'F'). Since you are creating plain text out of Caesar's messages, you will do the opposite:

Cipher text

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

Plain text

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

Only letters are shifted in this cipher. Any non-alphabetical character should remain the same, and all alphabetical characters will be upper case.

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets. All characters will be uppercase.

A single data set has 3 components:

1. *Start line* - A single line, "START"
2. *Cipher message* - A single line containing from one to two hundred characters, inclusive, comprising a single message from Caesar
3. *End line* - A single line, "END"

Following the final data set will be a single line, "ENDOFINPUT".

Output

For each data set, there will be exactly one line of output. This is the original message by Caesar.

Sample Input

```
START
NS BFW, JAJSYX TK NRUTYFSHJ FWJ YMJ WJXZQT TK YWNANFQ HFZXJX
END
START
N BTZQI WFYMJW GJ KNWXY NS F QNYYQJ NGJWNFS ANQQFLJ YMFS XJHTSI NS WTRJ
END
START
IFSLJW PSTBX KZQQ BJQQ YMFY HFJXFW NX RTWJ IFSLJWTZX YMFS MJ
END
ENDOFINPUT
```

Sample Output

```
IN WAR, EVENTS OF IMPORTANCE ARE THE RESULT OF TRIVIAL CAUSES
I WOULD RATHER BE FIRST IN A LITTLE IBERIAN VILLAGE THAN SECOND IN ROME
DANGER KNOWS FULL WELL THAT CAESAR IS MORE DANGEROUS THAN HE
```

Problem #2: Rubik's Cube Solver

Introduction

The Rubik's Cube was invented by Enro Rubik in 1974. It is a 3-dimensional puzzle made up of 26 smaller cubes. Each smaller cube has from one to three sides exposed for a total of 54 exposed sides. Each of these sides is assigned one of six colors, and each color is assigned to precisely nine exposed sides. The cube is manipulated by rotating any side of the cube by 90 degrees. It is considered solved when each side of the Rubik's Cube is entirely covered by one of the six colors.

You are a researcher at the Rubik's University and are working on an algorithm to solve a Rubik's Cube in the least possible number of moves from any starting position. To aid in your research, you need a program that will read in various Rubik's Cube configurations, perform operations on those configurations, and determine if the resulting cube is solved.

Input

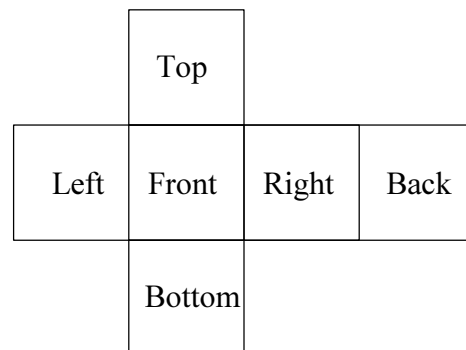
Input to this problem will consist of a starting configuration for the Rubik's Cube and then one or more operations to perform on the cube.

The input configuration will look like:

```

      G W O
      G R R
      G B R
B R B R G Y W W W Y G O
G W B O G B Y B O W Y O
W R Y O Y B R Y R G O O
      B R Y
      B O W
      G Y W
```

Which follows this layout:



Each character in this grid represents the color of the piece of the cube. There is one space between each character in the grid and possibly many spaces before the first character on a line. The grid represents the cube as is if it were unfolded and flattened out. Each group of 9 characters (3×3 array) represents one side of the grid. The top of the cube is represented by the first 3 lines of input. The next 3 lines of input represent the left, front, right, and back sides in that order. The last 3 lines represent the bottom of the cube.

Following the initial configuration will be 1 more operations to perform on the cube. There are 12 possible operations that can be performed, each being a 90 degree rotation of one of the cube's 'faces' of 9 smaller cubes. Note that this results in the movement of 20 colored squares (8 on the face being rotated and 12 on the sides of the smaller cubes that make up that face). All 12 possible operations are listed in the table below with a description of how to perform that operation.

Operation	Description
front left	Performed by rotating the front side counter-clockwise when viewing from the front
front right	Performed by rotating the front side clockwise when viewing from the front
left left	Performed by rotating the left side counter-clockwise when viewing from the left
left right	Performed by rotating the left side clockwise when viewing from the left
right left	Performed by rotating the right side counter-clockwise when viewing from the right
right right	Performed by rotating the right side clockwise when viewing from the right
back left	Performed by rotating the back side counter-clockwise when viewing from the back
back right	Performed by rotating the back side clockwise when viewing from the back
top left	Performed by rotating the top side counter-clockwise when viewing from the top
top right	Performed by rotating the top side clockwise when viewing from the top
bottom left	Performed by rotating the bottom side counter-clockwise when viewing from the bottom
bottom right	Performed by rotating the bottom side clockwise when viewing from the bottom

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets.

There will be one or more data sets. A single data set has 4 components:

1. *Start line* - A single line, "START".
2. An initial configuration of the cube (9 lines total)
3. One or more operations each on a separate line
4. *End line* - A single line, "END".

Following the final data set will be a single line, "ENDOFINPUT".

Output

For each data set, there will be exactly one line of output. If the cube is solved, then the word "Yes" is output on a line. If the cube remains unsolved, then the word "No" is output on a line.

Sample Input

```
START
    O O O
    O O O
    O G W
W W W Y B G R B B G G G
B Y W Y B G R G G W W R
B Y Y B O O B W W Y Y R
    R Y Y
    B R R
    G R R
front right
top right
left left
bottom right
END
START
    Y B B
    O G B
    G R B
W W B R Y Y R G G O O O
R O O G B R G R O Y W W
Y G Y O R G W B G O Y W
    B B R
    W Y Y
    R W W
back left
top right
left right
right right
front left
bottom right
right left
END
ENDOFINPUT
```

Sample Output

```
Yes
No
```

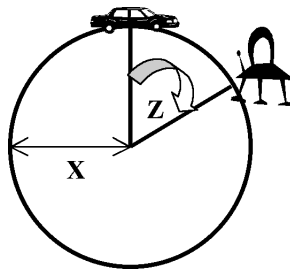
Problem #3: Polar Explorer

Introduction

You are a intrepid 2-dimensional explorer located at the northern polar reaches of a distant 2-dimensional planet. Unfortunately, you have been assigned to explore the most boring planet in the known universe (due primarily to your lack of social skills and aggressive body odor). Having a perfectly circular surface, your planet holds no surprises for a would-be explorer.

However, you have recently received a distress call from an alien ship which has crash-landed somewhere on the surface of your planet. Unfortunately, you designed your own equipment, and the only information it will give you is an angle (measured from the center of the planet) separating you from the crash site.

Using this information along with how much gasoline is available for your planet-rover (which gets a measley 5 miles per gallon), you have to determine if you can possibly get to the crash site and back without running out of fuel.



Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets.

A single data set has 4 components:

1. Start line - A single line, "START".
2. Input line - A single line, "X Y Z", where:
 3. X : (1 <= X <= 100) is the radius of your planet in integer miles
 - Y : (0 <= Y <= 100) is the amount of gasoline in your planet-rover in integer gallons
 - Z : (0 <= Z <= 360) is an angle separating you from the crash site in integer degrees
4. End line - A single line, "END".

Following the final data set will be a single line, "ENDOFINPUT".

Take note of the following:

- The circumference of a circle in terms of its radius, r, is known to be $2\pi r$
- Assume that $\pi = 3.14159$

Output

For each data set, there will be exactly one line of output. If you have enough fuel to get to the crash site and back, the line will read, "YES X", where X is the amount of fuel you will have left expressed as an integer number of gallons (truncate any fractional gallons). If you do not have sufficient fuel, the line will read, "NO Y", where Y is the distance you can travel expressed as an integer number of miles.

Sample Input

```
START
1 100 0
END
START
10 0 1
END
START
100 50 90
END
START
100 50 270
END
ENDOFINPUT
```

Sample Output

```
YES 100
NO 0
NO 250
NO 250
```

Problem #4: Door Man

Introduction

You are a butler in a large mansion. This mansion has so many rooms that they are merely referred to by number (room 0, 1, 2, 3, etc...). Your master is a particularly absent-minded lout and continually leaves doors open throughout a particular floor of the house. Over the years, you have mastered the art of traveling in a single path through the sloppy rooms and closing the doors behind you. Your biggest problem is determining whether it is possible to find a path through the sloppy rooms where you:

1. Always shut open doors behind you immediately after passing through
2. Never open a closed door
3. End up in your chambers (room 0) with all doors closed

In this problem, you are given a list of rooms and open doors between them (along with a starting room). It is not needed to determine a route, only if one is possible.

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets.

A single data set has 3 components:

1. *Start line* - A single line, "START M N", where M indicates the butler's starting room, and N indicates the number of rooms in the house ($1 \leq N \leq 20$).
2. *Room list* - A series of N lines. Each line lists, for a single room, every open door that leads to a room of higher number. For example, if room 3 had open doors to rooms 1, 5, and 7, the line for room 3 would read "5 7". The first line in the list represents room 0. The second line represents room 1, and so on until the last line, which represents room (N - 1). It is possible for lines to be empty (in particular, the last line will always be empty since it is the highest numbered room). On each line, the adjacent rooms are always listed in ascending order. It is possible for rooms to be connected by multiple doors!
3. *End line* - A single line, "END"

Following the final data set will be a single line, "ENDOFINPUT".

Note that there will be no more than 100 doors in any single data set.

Output

For each data set, there will be exactly one line of output. If it is possible for the butler (by following the rules in the introduction) to walk into his chambers and close the final open door behind him, print a line "YES X", where X is the number of doors he closed. Otherwise, print "NO".

Sample Input

START 1 2

1

END

START 0 5

1 2 2 3 3 4 4

END

START 0 10

1 9

2

3

4

5

6

7

8

9

END

ENDOFINPUT

Sample Output

YES 1

NO

YES 10

Problem #5: The Umbrella Problem: 2054

Introduction

"Forget it," Garret complained, throwing down the controller to his PlayStation VIII, "this level is impossible." He had just "died" for the 17th time on level 54 of the game "Lemmings 9: Lost in Space".

"No it isn't," his brother Ferret replied, "and I can prove it." Ferret pulled his PlaySkool PDA from the back pocket of his Levi's Huggies.

"First, picture the level as a rectangular grid." Ferret punched a few of the buttons on his PDA and a rectangle appeared as he described. "Your character, a Lemming holding an umbrella, starts at the top of this rectangle. His goal is to reach the bottom without dying."

"I know that, you weasel, but what about the laser guns?" Garret whined.

"The name is Ferret, and I was just getting to that. If we represent the level as a rectangular grid, then the Lemming can occupy one square and each laser gun can occupy a square. Remember the laser guns are cyclic: they all shoot up the first turn, right the second turn, down the third turn, left the fourth turn, and then repeat the sequence."

"But you're forgetting the pits of lava!" Garret exclaimed.

"You didn't let me finish. Each pit of lava also occupies a square. And each plot of grass, the Lemming's destination, can also occupy a square. Then, it's just a matter of manipulating the Lemming and laser beams in a series of turns to determine if it is possible for the Lemming to reach the bottom without 'dying'."

"You think you're so smart, Ferret, let's see if you can explain that again in a clear, concise way."

"Certainly":

The level will consist of a grid of squares. The way each laser beam and the Lemming moves can be described in "turns". To determine if the Lemming can reach the bottom of the level without dying, Ferret devised some rules:

1. Each turn will consist of two steps:
 - a. First, the laser guns will "fire" and maintain until the end of the turn, a beam in a direction dependent on the number of the turn. On the first turn, each laser gun will shoot up (all squares directly above a laser gun are "unsafe" and cannot be occupied by the Lemming); on the second turn, each laser gun will shoot right; on the third turn, each laser gun will shoot down; on the fourth turn, each laser gun will shoot left; on the fifth turn, the sequence will repeat.

Example:

```
Column
01234
R 0| L |<- The Lemming will always start in a column on row 0
o 1|   | In this example, on the first turn, the laser beam
w 2| S | will occupy squares (3,0), (3,1); second turn, (4,2);
  3|   | third turn, (3,3), (3,4), (3,5), (3,6); fourth turn,
  4|   | (0,2), (1,2), (2,2); fifth turn (repeating), (3,0), (3,1), etc.
  5|   | (squares are represented using (column,row) notation)
  6|GPPGG|<- The pits of lava and grass squares will always be
      in the last row
```

- b. Second, the Lemming will always move one row down, but to any one of three columns: one column to the left, one column to the right, or remain in the same column. In the above example, on the first turn the Lemming (L) could move to square (1,1), (2,1), or (3,1) (if he moved to

(3, 1), though, he would die because of the laser beam). However, on any turn the Lemming cannot move outside of the grid (i.e., he cannot move to column -1, or to a column number equal to the number of columns).

2. The level is considered "possible" if the Lemming can reach any "grass" square without dying after a series of turns.
3. The Lemming will die if at any point he occupies the same square as a laser gun, its beam, or a pit of lava. This includes:
 - a. The Lemming moving into a square a pit of lava occupies,
 - b. The Lemming moving into a square a laser gun occupies,
 - c. The Lemming moving into a square a laser beam occupies (even if it is a grass square!),
 - d. A laser gun firing a beam into a square the Lemming occupies

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets. Each data set will describe the starting conditions of the level. A single data set has the following components:

1. *Start line* - A single line, "START x y", where $0 < x < 10$ and x is the number of columns in the grid representing the level and $1 < y < 10$ and y is the number of rows in the grid representing the level.
2. The next y lines will represent the rows of the level, starting with row 0 (the top). Each line will consist of x letters. The letters will represent components of the level as follows:
 - L - Lemming (there will only be one 'L' per data set, and it will always be in row 0)
 - S - laser gun (these squares will never be in the final row)
 - P - pit of lava (these squares will always be in the final row)
 - G - grass (these squares will also always be in the final row)
 - O - "empty" square of air
3. *End line* -- A single line, "END".

Following the final data set will be a single line, "ENDOFINPUT".

Output

Output for each data set will be exactly one line. The line will either be "FERRET" or "GARRET" (both all caps with no whitespace leading or following).

"FERRET" will appear if the Lemming can make it safely (without dying) to any grass square at the bottom of the level after a series of turns.

"GARRET" will be output for a data set if it fails to meet the criteria for a "FERRET" line.

Sample Input

```
START 5 7
OOLOO
OOOOO
OOOSO
OOOOO
OOOOO
OOOOO
GPPGG
END
START 3 3
OLO
OSO
GGG
END
START 5 8
LOOOS
OOOOO
OOOOO
OOOOO
OOOOO
OOOOO
OOOOO
PPPPG
END
ENDOFINPUT
```

Sample Output

```
FERRET
GARRET
GARRET
```

Problem #6: Blue Gene, Jr

Introduction

Inspired by IBM's Blue Gene project, the CEO of Universal Biological Machinery (UBM), has called on you, UBM's top software engineer, to develop a program that will calculate the mutation of the Areopagus-virus, a virus discovered on Mars by your company's privately-subsidized (top-secret) space program.

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 3 components:

1. *Start line* - A single line, "START N", where $1 \leq N \leq 20$.
2. *Viral code* - A sequence of N alphanumeric characters. Alphanumeric characters will consist of an uppercase letter (A-Z) or a digit (0-9).
3. *End line* - A single line, "END"

Following the final data set will be a single line, "ENDOFINPUT".

Output

For each data set, there will be exactly one output set, and there will be no blank lines separating output sets.

A single output set consists of a single line of the viral code after it has stabilized (through mutating).

The viral code will mutate according to the following rules:

1. Initially the first viral segment to mutate begins with the first alphanumeric character of the viral code and ends with the rightmost alphanumeric character of the code.
2. If the first alphanumeric character of a viral segment is a letter (A-Z), that alphanumeric character is considered "unstable", and will mutate into n , where n is the number of mutations that occur to the viral segment immediately to the unstable alphanumeric character's right (see #5), unless n is greater than 9, in which case the unstable alphanumeric character will mutate into $(n \text{ modulo } 10)$. Also, if there is no viral segment immediately to the right of the unstable alphanumeric character, the unstable alphanumeric character will mutate into 0.
3. If the first alphanumeric character of a viral segment, n , is a positive number (1-9), that alphanumeric character is also considered "unstable", and will mutate into $n-1$. It also causes the viral segment beginning with the alphanumeric character n alphanumeric characters to its right and ending with the rightmost alphanumeric character of the viral code to mutate. If there is no alphanumeric character n alphanumeric characters to its right, then the viral segment immediately to its right (see #5), if one exists, will mutate.
4. If the first alphanumeric character of a viral segment is 0, that alphanumeric character is considered "stable", and will not mutate (the alphanumeric character will remain a 0 and a mutation will not be considered to have occurred).
5. A viral segment immediately to the right of an alphanumeric character begins with the alphanumeric character one position to its right and ending with the rightmost alphanumeric character of the viral code.

Sample Input

```
START 1
A
END
START 4
A1B2
END
START 15
A3B2CCC4AD1232R
END
START 15
0ABCDEFGHJKLMN
END
START 11
ABCDEFGHIJK
END
START 10
9AAAAAAAAA
END
ENDOFINPUT
```

Sample Output

```
0
3011
82B26543AD11310
0ABCDEFGHJKLMN
09876543210
8AAAAAAAAA0
```

Problem #7: Byte Me!

Introduction

You are a dealer at The One, the first all-binary casino in Las Vegas. What makes The One special is that its blackjack tables don't use cards; they use bytes (an 8-bit sequence representing a number from 0 to 255) and nibbles (a 4-bit sequence representing a number from 0 to 15).

All day long, you play the house's hand against individual opponents. Of course, the casino owners know their statistics, and they have devised a strategy for you that gives gamblers just less than even odds.

Here are the rules of binary blackjack:

1. The goal of the game is to be the player closest to 510 points without going over.
2. Each player is dealt two bytes, one face up and one face down.
3. The players then have the opportunity to take more bytes (by saying, "Byte Me!") or more nibbles (by saying, "Nibble Me!") until he reaches his limit of 4 hits or has more than 510 points showing.
4. All hits are played face up.
5. If a player goes over 510, he immediately *busts* and loses the hand.
6. The dealer hits last.
7. The dealer wins any ties (this includes a *tie* where everyone busts).

The rules for the dealer are (in order of precedence, where lower numbered rules override higher numbered ones):

1. Never hit when it is certain that you've won by simply looking at your hand and what is showing of other people's hands.
2. If your total is strictly less than 382 take a byte hit.
3. If your total is less than or equal to 500 take a nibble hit.
4. Take no hits

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be no blank lines separating data sets.

A single data set has 6 components:

1. *Start line*: A single line, "START N", where N is the number of players playing this hand (not including the dealer). There will never be more than 10 non-dealer players, and the dealer never plays by himself.
2. *Dealer Line*: A single line containing 2 binary strings of exactly eight digits separated by exactly one space. These two strings represent the two cards in the dealer's hand.
3. *Player Line*: A single line containing N 8-digit binary strings, each separated from the others by a single space. These represent the face-up cards of all of the non-dealer players.
4. *Byte Line*: A single line containing 4 8-digit binary strings, each separated from the others by a single space. These represent the next 4 bytes in the byte deck, in the order they will be drawn.
5. *Nibble Line*: A single line containing 4 4-digit binary strings, each separated from the others by a single space. These represent the next 4 nibbles in the nibble deck, in the order they will be drawn.
6. *End line*: A single line, "END".

Following the final data set will be a single line, "ENDOFINPUT".

Here are some other useful facts:

- Oddly enough, each non-dealer player is always dealt a face-down card 11111111, value 255, but the dealer has no knowledge of this.
- Players other than the dealer never hit (they aren't too bright).

Output

Calculate the actions taken by the dealer and how the dealer fares with the resulting hand.

For each data set, there will be exactly one output set, consisting of the following components:

1. *Hand Line*: A single line, "HAND N", where N is the number of players playing this hand (not including the dealer).
2. *Dealer Hit List*: A single line will be printed for each hit the dealer takes on his turn. For a byte hit, print a line "Byte me!", and for a nibble hit print, "Nibble me!"
3. *Outcome Line*: A single line containing "Win!" if the dealer wins, "Bust!" if the dealer loses by busting, and "Lose!" if the dealer loses without busting.

Sample Input

```
START 1
11111111 11111111
00000001
10101010 01010101 11110000 00001111
1010 0101 1100 0011
END
START 1
10111110 10111111
11111110
00010010 10101010 01010101 11110000
0001 1010 1100 0011
END
START 8
11111111 00001000
00000000 00000001 00000010 00000011 00000100 00000101 00000110 00000111
00010010 10101010 01010101 11110000
0001 1010 1100 0011
END
ENDOFINPUT
```

Sample Output

```
HAND 1
Win!
HAND 1
Byte me!
Nibble me!
Nibble me!
Nibble me!
Lose!
HAND 8
Win!
```


Problem #8: World's Worst Bus Schedule

Introduction

You are a very impatient person, and hate to be kept waiting. You are on your way to visit a relative all the way in New Orleans. The problem is that the bus station you are at has the world's worst bus schedule! There are no arrival or departure times listed, only route durations for each bus running. Being the impatient person you are, you whip out your laptop and attempt to write a program that will determine how long you will have to wait before the next bus comes. Hey, you have nothing better to do, right?

Input

Input to this problem will consist of a (non-empty) series of up to 100 data sets. Each data set will be formatted according to the following description, and there will be **no blank lines** separating data sets.

A single data set has 4 components:

1. *Start line* - A single line, "START N", where N is the number of buses running and $1 \leq N \leq 20$.
2. *Route Duration line* - There will be N of these lines, and each line will consist of M route durations (where $1 \leq M \leq 10$), which will signify how long it will take each bus to return to the bus station after completing a particular route. A route duration will be represented as an integer between 1 and 1000 (inclusive).
3. *Arrival Time* - There will only be one of these lines per data set. This line represents the time that you arrived at the bus station and began waiting. This is simply the number of time units that you arrived at the bus station after the buses began running (all buses begin running at time 0). This number is represented as a positive integer (yes, it can be 0 as well, this would signify arriving at the bus station as the buses begin running).
4. *End line* - A single line, "END".

Following the final data set will be a single line, "ENDOFINPUT".

Output

For each data set, there will be exactly one line of output. This line will simply be the number of time units you will have to wait before the next bus comes after you arrive. You hate waiting so much that you will just get on the first bus that returns to the station. Let's hope this is the bus to New Orleans!

Notes

1. All buses continuously go on their routes, starting back up with their first route after their last route is done.
2. If the passenger's arrival time coincides with any of the bus' route departure times, he/she catches the bus at that time.

Sample Input

```
START 3
100 200 300
400 500 600
700 800 900
1000
END
START 3
100 200 300 4 3 2 4 2 22
800
10 1000
32767
END
ENDOFINPUT
```

Sample Output

```
200
20
```