Problem A: Rock, Paper, or Scissors?

Rock, Paper, Scissors is a two player game, where each player simultaneously chooses one of the three items after counting to three. The game typically lasts a pre-determined number of rounds. The player who wins the most rounds wins the game. Given the number of rounds the players will compete, it is your job to determine which player wins after those rounds have been played.

The rules for what item wins are as follows:

- Rock always beats Scissors (Rock crushes Scissors)
- Scissors always beat Paper (Scissors cut Paper)
- Paper always beats Rock (Paper covers Rock)

Input (from file a.in)

The first value in the input file will be an integer **`t`** (**`0 < t < 1000`**) representing the number of test cases in the input file. Following this, on a case by case basis, will be an integer **`n (0 < n < 100)`** specifying the number of rounds of Rock, Paper, Scissors played. Next will be **`n`** lines, each with either a capital R, P, or S, followed by a space, followed by a capital R, P, or S, followed by a newline. The first letter is Player 1's choice; the second letter is Player 2's choice.

Output (to monitor)

For each test case, report the name of the player (Player 1 or Player 2) that wins the game, followed by a newline. If the game ends up in a tie, print TIE.

Sample Input

```
3
2
R P
S R
3
P P
R S
S R
1
P R
```

Sample Output

```
Player 2
TIE
Player 1
```

Problem B: Missile Command

As Chief Bureaucrat at Missile Command, it has recently come to your attention that the existing performance guidelines do not sufficiently penalize frivolous use of expensive ammunition. Therefore, you must write a new battle summary analysis tool which takes into account excess ammunition consumption during battle.

A battle consists of the following elements:

- Shots. A shot is a circularly explosive countermeasure. A shot has a fixed position and is active for 2 seconds, during which its radius varies from 0 to 1km and then back to 0 according to the formula:
  $r = (1 - (t - 1)^2)^{1/2}$
- The ground, at y = 0.
- Missiles. A missile is a point particle that moves at a constant velocity. If a missile collides with an active shot, the missile is neutralized (the shot persists). If a missile hits the ground before being neutralized, it is considered to have hit its target.

Performance is evaluated on a simple point scale. The performance criteria are as follows:

- Every neutralized missile adds 1 point.
- Every missile allowed to hit its target subtracts 5 points.
- Every unnecessary shot subtracts 20 points. The number of unnecessary shots in a battle is the difference between the actual number of shots fired and size of the minimum subset of those shots that would have neutralized the same number of missiles.

Input (from file b.in)

Input will be given in the following format (legend follows):

```
nb
nm
mx my mdx mdy mt
...
ns
sx sy st
...
...
```

In the following legend, indentation denotes repetition of the indented block a number of times equal to the value of the preceding input item:

nb (0 < *nb*) – number of battles
    nm (0 <= *nm* <= 20) – number of missiles
        mx/*my* (0.0 < *my*) – initial missile position (in km)
        mdx/*mdy* – missile velocity (in km/s)
        mt (0.0 <= *mt*) – time since battle start of the missile's entrance (in seconds)
    ns (0 <= *ns* <= 20) – number of shots
        sx/*sy* (1.0 <= *sy*) – shot position at time of detonation (in km)
        st (0.0 <= *st*) – time since battle start of the shot's detonation (in seconds)

Output (to stdout)

For each battle, output a line containing the score for that battle.

Sample Input

```
2
2
4.0 8.0 0.0 -1.0 0.0
4.0 8.0 1.0 -1.0 0.0
1
4.0 4.0 3.0
3
4.0 10.0 0.0 -1.0 0.0
5.0 10.0 3.0 -6.0 4.0
13.0 10.0 -3.0 -5.0 4.0
3
4.0 5.0 3.0
7.0 8.0 4.0
9.0 4.0 4.0
```

Sample Output

```
-4
-17
```

Problem C: Palindromic Primes Category in Jeopardy!

Prime numbers are defined as follows: a number is prime if it is greater than 1 and is evenly divisible only by itself and 1. Note that by definition neither zero nor one is a prime number.

A palindromic number is one whose string representation is a palindrome, that is, a string that reads the same backwards and forwards.

You are on the clue crew preparing questions for the category "Palindromic Primes" and are to write a program to generate the answer and responding question in Jeopardy! style.

Input (from file c.in)

The input file contains a series of number pairs (with white space separating them) specifying individual problems, ending with a pair of zeroes. The first number gives the number of digits for the numbers to be considered, the second number gives the base in which the numbers are to be generated. The numbers are separated by a single space. You are assured that all palindromic primes for this problem can be represented in the range of a standard 32-bit signed integer. The bases allowed are integer bases between 2 and 36 — with bases above base ten handled as extensions of hexadecimal. This means that the valid numeric digits are in the range `['0'..'9']` and `['a'..'z']`.

Output (to monitor)

For each number, generate one line giving the number of digits and the base as the answer and then on the next line the number of palindromic primes found as the question as shown in the sample output. Each output pair should be separated by a blank line.

Sample Input

```
1 10
2 10
3 10
4 24
5 4
0 0
```

Sample Output

```
The number of 1-digit palindromic primes < 2^31 in base 10.
What is 4?

The number of 2-digit palindromic primes < 2^31 in base 10.
What is 1?

The number of 3-digit palindromic primes < 2^31 in base 10.
What is 15?

The number of 4-digit palindromic primes < 2^31 in base 24.
What is 0?

The number of 5-digit palindromic primes < 2^31 in base 4.
What is 10?
```

Problem D: Pebbles

You're given an unlimited number of pebbles to distribute across an N x N game board (N drawn from [3, 15]), where each square on the board contains some positive point value between 10 and 99, inclusive. A 6 x 6 board might look like this:

33	**74**	**26**	**55**	**79**	**54**
67	**56**	**91**	**72**	**44**	**32**
44	**64**	**22**	**91**	**29**	**61**
61	**32**	**76**	**50**	**50**	**32**
81	**65**	**56**	**38**	**96**	**36**
38	**78**	**50**	**92**	**90**	**75**

The player distributes pebbles across the board so that:

- At most one pebble resides in any given square.
- No two pebbles are placed on adjacent squares. Two squares are considered adjacent if they are horizontal, vertical, or even diagonal neighbors. There's no board wrap, so 44 and 61 of row three aren't neighbors. Neither are 33 and 75 nor 55 and 92.

The goal is to maximize the number of points claimed by your placement of pebbles.

Write a program that reads in a sequence of boards from an input file and prints to stdout the maximum number of points attainable by an optimal pebble placement for each.

Input (from file d.in)

Each board is expressed as a series of lines, where each line is a space-delimited series of numbers. A blank line marks the end of each board (including the last one)

Output (to monitor)

then your program would print the maximum number of points one can get by optimally distributing pebbles while respecting the two rules, which would be this (each output should be printed on a single line and followed with a newline):

Sample Input

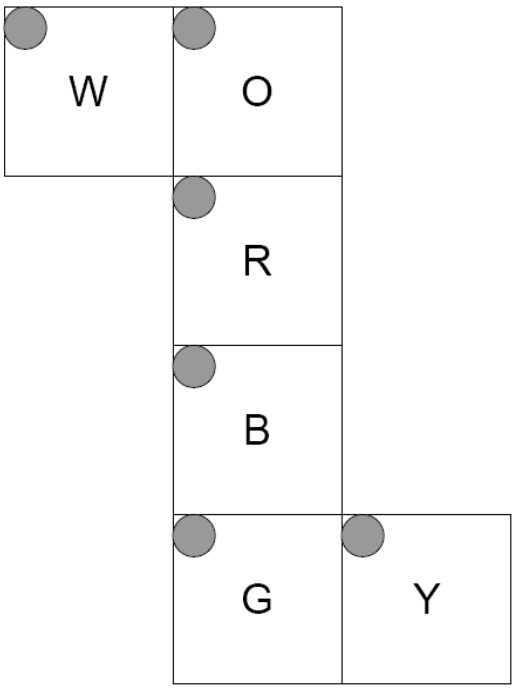
```
71 24 95 56 54
85 50 74 94 28
92 96 23 71 10
23 61 31 30 46
64 33 32 95 89

78 78 11 55 20 11
98 54 81 43 39 97
12 15 79 99 58 10
13 79 83 65 34 17
85 59 61 12 58 97
40 63 97 85 66 90

33 49 78 79 30 16 34 88 54 39 26
80 21 32 71 89 63 39 52 90 14 89
49 66 33 19 45 61 31 29 84 98 58
36 53 35 33 88 90 19 23 76 23 76
77 27 25 42 70 36 35 91 17 79 43
33 85 33 59 47 46 63 75 98 96 55
75 88 10 57 85 71 34 10 59 84 45
29 34 43 46 75 28 47 63 48 16 19
62 57 91 85 89 70 80 30 19 38 14
61 35 36 20 38 18 89 64 63 88 83
45 46 89 53 83 59 48 45 87 98 21

15 95 24 35 79 35 55 66 91 95 86 87
94 15 84 42 88 83 64 50 22 99 13 32
85 12 43 39 41 23 35 97 54 98 18 85
84 61 77 96 49 38 75 95 16 71 22 14
18 72 97 94 43 18 59 78 33 80 68 59
26 94 78 87 78 92 59 83 26 88 91 91
34 84 53 98 83 49 60 11 55 17 51 75
29 80 14 79 15 18 94 39 69 24 93 41
66 64 88 82 21 56 16 41 57 74 51 79
49 15 59 21 37 27 78 41 38 82 19 62
54 91 47 29 38 67 52 92 81 99 11 27
31 62 32 97 42 93 43 79 88 44 54 48
```

Sample Output

```
572
683
2096
2755
```

Problem E: Rubik's Cube®

You are a secret agent for the Eternally Indebted External Intelligence Office (EIEIO) of the country of Nomoneo. Headquarters has disguised your one-time pad for encrypted communications in the form of a Rubik's Cube®. (For those of you unfamiliar with the puzzle, a Rubik's Cube® comes in the form of a cube where each face is divided into three rows and three columns (nine "squares"). Any of the six faces of the cube may be rotated either clockwise or counterclockwise, which also rotates the three nearest squares on each adjoining face onto a new face, respectively. When solved (or taken from the factory packaging), each face of the cube contains squares of only one color. There is no way to change the relationship between the colors of the central squares on each face.) The cube has been pre-scrambled and you are to apply a certain set of moves to the cube based on the message you want to return.



This diagram provides the relationship between the sides of the cube as well as the orientation of the faces for the purposes of input and output. It should be viewed as an unfolded cube with the text on the outside. The faces are indicated by the color of the central subdivision (square) and are White, Orange, Red, Blue, Green, and Yellow. The corner with the dot is the top left corner for purposes of input and output.

Input (from file e.in)

Your program should read the input data from the file, which consists of several cases. The first line of the input will give the number of these input cases (as a decimal integer without any punctuation), which will be greater than or equal to 1 and less than or equal to 10,000. Each case consists of three lines giving the initial state of the puzzle cube and one line giving the rotations you must apply to reach the appropriate final state of the cube.

Each of the three lines giving the initial state of the cube consists of 18 letters with a single space between each pair of letters. There is no space between the last letter and the end of the line. Each of the letters is 'W', 'O', 'R', 'B', 'G', or 'Y' indicating the respective color. The faces are listed from left to right in the order (by central square color) White, Orange, Red, Blue, Green, Yellow. Thus, the first three columns of letters give the state of the "White" face, and so forth.

The fourth line of the case gives the manipulations that you must apply to the cube. Each manipulation consists of a single letter as above ('W', 'O', 'R', 'B', 'G', or 'Y') indicating which face (selected by the color of the center) you should rotate. Each rotation thus indicated is a 90º clockwise rotation of the face with respect to the rest of the cube, looking at the face to be rotated. At least one and no more than 1,000 manipulations will be specified.

Output (to monitor)

Print to standard output the state of the cube after the indicated manipulations. Use the same format as for input: three lines, each containing 18 color letters separated by a single space character. Do not put a space after the final letter on the line. Print out the faces in the same order, left to right, as for input: White, Orange, Red, Blue, Green, Yellow. After each output case, print a line containing 35 '=' characters.

Sample Input

```
2
W W W O O O R R R B B B G G G Y Y Y
W W W O O O R R R B B B G G G Y Y Y
W W W O O O R R R B B B G G G Y Y Y
RG
Y G G Y G W Y W B R W O R W G B G G
O W R Y O B G R O R B O Y G R B Y Y
O W O G O W Y B W B R W B B O R Y R
GROWBOBGROW
```

Sample Output

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

Problem F: Geek Challenge [SKRZAT] (Base Minus Two)

Geek Challenge [SKRZAT] is an **old**, **old** game from Poland that uses a game console with two buttons plus a joy stick. As is true to its name, the game communicates in binary, so that one button represents a zero and the other a one. Even more true to its name, the game chooses to communicate so that the base of the number system is **minus two**, not plus two, so we'll call this representation "Weird Binary". Thus the bit positions label the powers of **minus two**, as seen in the following five-bit tables:

Bits	Value	Bits	Value	Bits	Value	Bits	Value
00000	0	01000	−8	10000	16	11000	8
00001	1	01001	−7	10001	17	11001	9
00010	−2	01010	−10	10010	14	11010	6
00011	−1	01011	−9	10011	15	11011	7
00100	4	01100	−4	10100	20	11100	12
00101	5	01101	−3	10101	21	11101	13
00110	2	01110	−6	10110	18	11110	10
00111	3	01111	−5	10111	19	11111	11

Bits	Value	Bits	Value	Bits	Value	Bits	Value
01010	−10	00010	−2	11010	6	10010	14
01011	−9	00011	−1	11011	7	10011	15
01000	−8	00000	0	11000	8	10000	16
01001	−7	00001	1	11001	9	10001	17
01110	−6	00110	2	11110	10	10110	18
01111	−5	00111	3	11111	11	10111	19
01100	−4	00100	4	11100	12	10100	20
01101	−3	00101	5	11101	13	10101	21

Numbers are presented on the screen in Weird Binary, and then numbers are accepted in response from the console as a stream of zeroes and ones, terminated by a five-second pause.

You are writing a computer program to support the novice geek in playing the game by translating numbers between decimal and Weird Binary.

Input (from file f.in)

The first line in the file gives the number of problems being posed without any white space. Following are that many lines. Each line will either be a conversion into Weird Binary or out of Weird Binary: the letter "b" indicates that the rest of the line is written in Weird Binary and needs to be converted to decimal; the letter "d" indicates that the rest of the line is written in decimal and needs to be converted to Weird Binary.

The input data are in the range to fit within a 15-bit Weird Binary number, which represents the decimal number range –10922 to 21845, inclusive.

Output (to monitor)

For each conversion problem, show the type of problem, its input string, and the converted result in the format shown below, replicating even the spacing exactly as shown. Leading zeroes are not allowed.

Sample Input

```
10
b 1001101
b 0111111
b 101001000100001
b 010010001000010
b 100110100110100
d -137
d 137
d 8191
d -10000
d 21000
```

Sample Output

```
From binary: 1001101 is 61
From binary: 0111111 is -21
From binary: 101001000100001 is 19937
From binary: 010010001000010 is -7106
From binary: 100110100110100 is 15604
From decimal: -137 is 10001011
From decimal: 137 is 110011001
From decimal: 8191 is 110000000000011
From decimal: -10000 is 10100100110000
From decimal: 21000 is 101011000011000
```

Problem G: Frogger

Philip J. Frog just wanted to go for a mid-afternoon swim, but in typical frog fashion he's ended up in the middle of a busy street. Help Phil figure out how long he'll be hopping on hot asphalt before he finds his way to the nice cool water.

Phil may hop one square horizontally or vertically per second. He may only hop onto road, grass, or water. Additionally, he cannot occupy any square occupied by a car. Phil and the cars move at the same time, meaning Phil can "hop over" an oncoming car. Phil can also remain in the same square if he wishes. All horizontal movement wraps (e.g., a rightward hop from the rightmost column places Phil in the leftmost column). Cars move horizontally in the direction indicated on the map ('<' means leftward, '>' means rightward) at a rate of one square per second and never collide with anything.

Input (from file g.in)

Input begins with a single integer specifying the number of test maps. Each map begins with two integers R and C (`0 < R, C <= 30`) specifying the number of rows and columns, respectively, followed by R lines each C characters long, specifying the map. The possible map characters are:

Phil ('`&`') - Phil's starting location. Each map contains exactly one. Always indicates road underneath.
Tree ('`T`') - Impassable.
Grass ('`.`') - Phil can move freely in the grass.
Road ('–') - Hot!
Car ('<', '>') - Always indicates road underneath.
Water ('~') - Phil's goal.

Output (to monitor)

For each map, output a line containing the fewest number of seconds Phil must spend on the road in order to reach the water, or the string "`Impassable`", if no path to water exists.

Sample Input

```
3
2 1
~
&
4 7
~TTTTTT
.------
-->-<--
---&---
3 5
~~~~~
..T..
>>&<<
```

Sample Output

```
1
6
Impassable
```

Problem H: "Bubble Gum, Bubble Gum, in the dish, how many pieces do you wish?"

Alex and Karyn were at it again. The elementary school sisters were playing their favorite game to decide who gets to play on the computer next.

The rules of the game are quite simple. Given **p** people (**p > 0**), one of the **p** people is chosen to pick a number **n** (**n > p**) representing the number of pieces of bubble gum desired. Once this value is chosen, the people are iterated through, one at a time, starting at 1, from "left" to "right", starting with the person who chose the number. Iterating is done in a circular fashion, meaning that once the person on the far right is reached, the next person in the iteration will be the person on the far left. Upon reaching **n**, the person at that location is the winner.

Given a list of names, followed by the name of the person choosing the number of pieces of bubble gum, followed by the number that person chose, determine who wins the game.

Input (from file h.in)

The first value in the input file will be an integer **t** (**0 < t < 1000**) representing the number of test cases in the input file. Following this, on a case by case basis, will be a list of the names of the people (**p**), on a single line. Names will be no larger than 20 characters in length and all names are unique. There will be no more than 20 names. Each name is followed by a space, save for the last name, which is followed by a newline. On the next line is the name of the person choosing the number of pieces of bubble gum, followed by a newline. The test case is concluded with the number of pieces of gum **n** (**p < n < 1000**), which is also followed by a newline.

Output (to monitor)

For each test case, report the name of the person that won the game, followed by a newline.

Sample Input

```
3
Alex Karyn Maude
Karyn
5
Alex Karyn Maude
Alex
6
Alex Karyn Zach Becca Maude
Zach
8
```

Sample Output

```
Maude
Maude
Maude
```

Problem I: Games R Us

GamesAreUs.com has just completed its outside audit for this year. One item that was caught was the lack of any business rules for assigning permissions to files on the company's shared file server. The analysts are working on setting up some roles for all employees and what permissions should be given to each role. Your team is to take a look at the existing situation so you can provide some input to the analysts.

Fortunately permission assignment has not been completely random. The most common way to set up a new employee is to ask that they be set up "just like Joe", effectively making an ad-hoc prototype system.

You will be given the access control lists (ACLs) for the top level directories of the shared file server. Using these, your team is to write a program to split the users up into equivalence classes where all members of a class have access to exactly the same directories.

Because of the several departments in GamesAreUs.com, there are multiple access control lists to be processed: one set of lists per department.

Input (from file i.in)

The first line of input to your program is a single integer **n** (**0 < n < 100**), by itself on the line without any white space, giving the number of departments. Following that are the data for those departments.

ACLs associated with one department is a sequence of ACLs ended by "-1" by itself. Each ACL is a line of unsigned integers (`1 <= x <= 2147483647`) separated from each other by a single blank. The first integer on the line is the file id (FID) of the directory. The remaining numbers on the line are the user ids (UIDs) that have access. Each line will have a FID and at least one UID. There will be no duplicate UIDs on the line, but note the UIDs and FIDs are in separate spaces so a UID could be the same integer as a FID. The ACLs, and within an ACL the UIDs, appear in no particular order. In the full list of ACLs, a given FID will appear only once. There are at most 50 top-level directory ACLs and there are at most 100 UIDs. All FIDs and UIDs are greater than 0.

Output (to monitor)

For each department, the first line of output identifies which case is being processed, beginning with 1. That line contain the word "`Case`", one blank, and the integer identifying which case it is.

For every class with at least 2 members, print a line with the number of members in the class with no sign or leading zeros, one space, and the smallest UID in the class with no sign, leading zeros, or trailing spaces. Sort the output by the number of members, descending, and then by the UIDs, ascending. If there are no such classes, print "`no prototypes found`".

Sample input

```
2
100 9 7 2 3 1 6
200 9 6 7 1 2 5 3 8
300 6 3 7 8 5
400 4 5 8
-1
100 1
200 37
-1
```

Sample output

```
Case 1
3 1
3 3
2 5
Case 2
no prototypes found
```

Problem J: Cover Up

In the Price Is Right's game Cover Up, players test their luck to win a new car. To win, the player must produce the actual retail price of the car from a board of possible numbers like:

```
          9
        3 4
      0 4 7
    9 6 7 3
1 4 8 2 6
3 2 4 0 8
```

The player selects one number from each column to form a *bid*. Using the above board as an example, the first number in the price of the car is either 1 or 3; the second is one of 9, 4, 2; the third is one of 0, 6, 8, 4; and so on. Numbers may never move to a different column.

After the player selects their bid, Drew Carey lights up the numbers in the bid which are correct. If the player has no numbers correct, the game ends and they lose; if they have at least one number correct, the game continues.

When the game continues, the player is given another opportunity to select numbers from those columns that were incorrect. They will *cover up* the wrong bid numbers with different selections from the same columns. Again, Drew Carey will light up any new correct digits. If the player has no *new* numbers correct, the game ends and they lose; if they have at least one new number correct, the game continues.

For example:

```
          9                 9                 9
        3 4               3 4               3 4
      0 4 7             0 4               0 4
    9 6 7 3    ->     9     3     ->    9
1 4 8 2 6               8 2 6                 2 6
3 2 4 0 8         3 2 4 0 8           3 2 4 0 8
| | | | |             |   |               |
v v v v v             v   v               v

                  1 4 6 7 7           1 4 8 7 3
                  c c x c x           c c x c c

                  INITIAL BID         SECOND BID
```

The player selects an initial bid of $14677. The 1, 4 and first 7 are correct (**c** stands for correct, **x** for incorrect, and **v** designates a column that requires a subsequent selection in the example above). The player covers up the incorrect 6 and 7 with an 8 and a 3 for a second bid of $14873. The 3 is correct, but the 8 is wrong. At this point it's a 50/50 chance. The player will select the 4 or the 0 and either win the car or lose the game.

The show's sponsors would like to know how frequently their cars are given away. You are to use the assumption that players choose numbers uniformly from those remaining.

Input (from file j.in)

The sponsor will explore many variations, with prices up to 7 digits long. Therefore, the input file will begin with a line containing the integer **N <= 5000**, the number of test cases to be explored. The test cases follow.

Each test case begins with the integer **d**, **0 < d <= 7**, the number of digits in the price of the car. **d** lines will follow, with non-empty strings of *distinct* digits in the range from 0 through 9. Each of these lines represents a *column* of the digits in the game. The first line represents the leftmost column; the last the rightmost column. A 0 is possible as the first digit in the price of the car.

Output (to monitor)

Your program will print the probability of the player winning the car, rounded to 3 decimals.

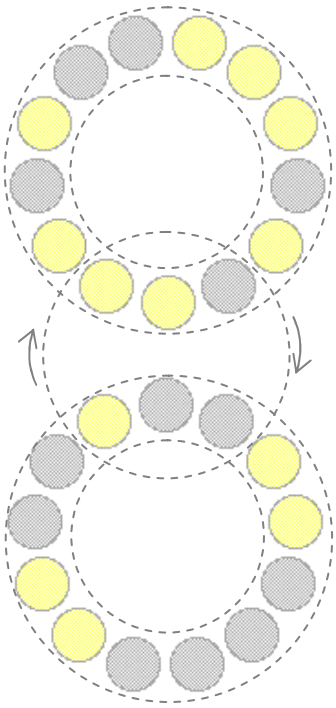
Sample Input

```
2
2
9
19
5
13
942
0684
34720
947368
```

Sample Output

```
1.000
0.321
```

Problem K: Bridged Marble Rings



26 marbles—half yellow and half gray—are distributed between two circles of 13 marbles each. The marbles in each circle can be freely rotated clockwise or counterclockwise. The upper and lower circles are bridged by a smaller circle, which rotates—in the plane of the board—180 degrees, effectively exchanging the three bottommost marbles of the upper circle with the three uppermost marbles of the lower one. The goal is to get all gray marbles to the upper circle and all yellow marbles to the lower one while minimizing the number of times the bridging circle is rotated.

Input (from file k.in)

The input is a series of lines, where each line describes an initial board configuration. Each line is a permutation of 13 **y**'s and 13 **g**'s. The first half of the line describes the clockwise configuration of the upper circle, and the rest of the line describes the clockwise configuration of the lower one. Of course, each **y** corresponds to a yellow marble, and each **g** corresponds to a gray one.

The input file will include multiple test cases. Each test case consists of a single line containing some permutation of the string $\mathbf{y}^{13}\mathbf{g}^{13}$. All lines (including the last one) are terminated with a newline. The newline immediately follows the last letter on the line.

Output

For each input case, you should print the minimum number of bridge rotations on a single line.

Sample Input

```
gggggggggggggyyyyyyyyyyyyy
yyyyyggggggggyyyyyggggggyyy
gyyygyggyyygyygggyyggggyygg
ygygygygygygygygygygygygygyg
```

Sample Output

```
0
2
5
6
```