The Seventh Hunan Collegiate Programming Contest

17th September, 2011
You get 12 Pages
11 Problems
&
300 Minutes

A One-Two-Three

Input: Standard Input
Output: Standard Output

Your little brother has just learnt to write one, two and three, in English. He has written a lot of those words in a paper, your task is to recognize them. Note that your little brother is only a child, so he may make small mistakes: for each word, there might be at most one wrong letter. The word length is always correct. It is guaranteed that each letter he wrote is in lower-case, and each word he wrote has a unique interpretation.

Input

The first line contains the number of words that your little brother has written. Each of the following lines contains a single word with all letters in lower-case. The words satisfy the constraints above: at most one letter might be wrong, but the word length is always correct. There will be at most 10 words in the input.

Output

For each test case, print the numerical value of the word.

Sample Input	Output for Sample Input
3 owe too theee	1 2 3

Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

B Counting Game

Input: Standard Input
Output: Standard Output

There are n people standing in a line, playing a famous game called "counting". When the game begins, the leftmost person says "1" loudly, then the second person (people are numbered 1 to n from left to right) says "2" loudly. This is followed by the 3rd person saying "3" and the 4th person say "4", and so on. When the n-th person (i.e. the rightmost person) said "n" loudly, the next turn goes to his immediate left person (i.e. the (n-1)-th person), who should say "n+1" loudly, then the (n-2)-th person should say "n+2" loudly. After the leftmost person spoke again, the counting goes right again.

There is a catch, though (otherwise, the game would be very boring!): if a person should say a number who is a multiple of 7, or its decimal representation contains the digit 7, he should clap instead! The following tables shows us the counting process for n=4 ('X' represents a clap). When the 3rd person claps for the 4th time, he's actually counting 35.

Person	1	2	3	4	3	2	1	2	3
Action	1	2	3	4	5	6	X	8	9
Person	4	3	2	1	2	3	4	3	2
Action	10	11	12	13	X	15	16	X	18
Person	1	2	3	4	3	2	1	2	3
Action	19	20	X	22	23	24	25	26	X
Person	4	3	2	1	2	3	4	3	2
Action	X	29	30	31	32	33	34	X	36

Given n, m and k, your task is to find out, when the m-th person claps for the k-th time, what is the actual number being counted.

Input

There will be at most 10 test cases in the input. Each test case contains three integers n, m and k (2<=n<=100, 1<=m<=n, 1<=k<=100) in a single line. The last test case is followed by a line with n=m=k=0, which should not be processed.

Output

For each line, print the actual number being counted, when the m-th person claps for the k-th time. If this can never happen, print `-1`.

Sample Input	Output for Sample Input
4 3 1 4 3 2 4 3 3 4 3 4 0 0 0	17 21 27 35

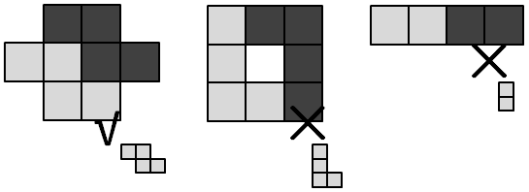
Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

C Polyomino Composer

Input: Standard Input
Output: Standard Output

A polyomino is a plane geometric figure formed by joining one or more equal squares edge to edge.
-- Wikipedia

Given a large polyomino and a small polyomino, your task is to determine whether you can compose the large one with two copies of the small one. The polyominoes can be translated, but not flipped or rotated. The two pieces should not overlap. The leftmost picture below is a correct way of composing the large polyomino, but the right two pictures are not. In the middle picture, one of the pieces was rotated. In the rightmost picture, both pieces are exactly identical, but they're both rotated from the original piece (shown in the lower-right part of the picture).



Input

There will be at most 20 test cases. Each test case begins with two integers n and m ($1<=m<=n<=10$) in a single line. The next n lines describe the large polyomino. Each of these lines contains exactly n characters in {'*','.'}. A '*' indicates an existing square, and a '.' indicates an empty square. The next m lines describe the small polyomino, in the same format. These characters are guaranteed to form valid polyominoes (note that a polyomino contains at least one existing square). The input terminates with $n=m=0$, which should not be processed.

Output

For each case, print 1 if the corresponding composing is possible, print 0 otherwise.

Sample Input

```
4 3
.**.
****
.**.
....
**.
.**
...
3 3
***
*.*
***
*..
*..
**.
4 2
****
....
....
....
*.
*.
0 0
```

Output for Sample Input

```
1
0
0
```

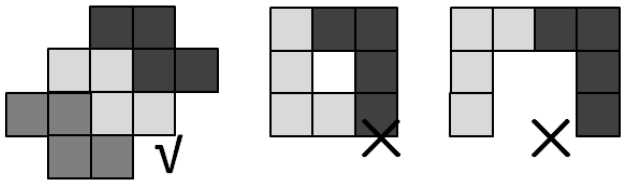
Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

D Polyomino Decomposer

Input: Standard Input
Output: Standard Output

A polyomino is a plane geometric figure formed by joining one or more equal squares edge to edge.
-- Wikipedia

Given a large polyomino, your task is to decompose it into at least two identical polyominoes (can't be flipped or rotated). The leftmost picture below is a correct way of decomposing, but the right two pictures are not. In the middle picture, one of the pieces is rotated. In the rightmost picture, one of the pieces is flipped. The number of pieces should be as small as possible. Note that there is at least one solution: decompose it into a lot of unit squares.



Input

There will be at most 30 test cases. Each test case begins with an integer n ($1<=n<=10$) in a single line. The next n lines describe the large polyomino. Each of these lines contains exactly n characters in {'*','.'}. A '*' indicates an existing square, and a '.' indicates an empty square. These characters are guaranteed to form a valid polyomino. There are at least one and at most twenty existing squares in the large polyomino. The input terminates with n=0, which should not be processed.

Output

For each test case, output a line containing the decomposition of the large polyomino. Each existing square is replaced by a capital letter representing the label of the piece. Different pieces should use different labels. If there are multiple solutions, print the lexicographically smallest one. That is, if we regard the solution as a long string, by concatenating rows from top to bottom, your output should be lexicographically the smallest one among all possible solutions. Print an empty line after each case.

Sample Input

```
5
..**.
.****
****.
.**..
.....
2
**
**
0
```

Output for Sample Input

```
..AA.
.AABB
AABB.
.BB..
.....

AA
BB
```

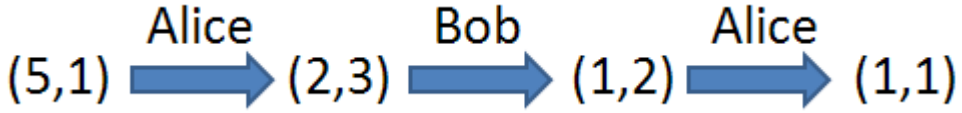
Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

E Box Game

Input: Standard Input
Output: Standard Output

There are two identical boxes. One of them contains *n* balls, while the other box contains one ball. Alice and Bob invented a game with the boxes and balls, which is played as follows:

Alice and Bob moves alternatively, Alice moves first. For each move, the player finds out the box having fewer number of balls inside, and empties that box (the balls inside will be removed forever), and redistribute the balls in the other box. After the redistribution, each box should contain at least one ball. If a player cannot perform a valid move, he loses. A typical game is shown below:



When both boxes contain only one ball, Bob cannot do anything more, so Alice wins.

Question: if Alice and Bob are both clever enough, who will win? Suppose both of them are very smart and always follows a perfect strategy.

Input

There will be at most 300 test cases. Each test case contains an integer n ($2<=n<=10^9$) in a single line. The input terminates by *n*=0.

Output

For each test case, print a single line, the name of the winner.

Sample Input	Output for Sample Input
2 3 4 0	Alice Bob Alice

Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

F RPG battles

Input: Standard Input
Output: Standard Output

In many typical RPG games, you battle with bad guys, creatures, monsters or ghosts etc. all the time. After each battle, you may get magic potions that power you up, so you'll get stronger and stronger, and finally defeat the big boss. In this problem, we only consider two kinds of potion: *stronger* and *double power*. If you drink a bottle of *stronger potion*, your power increases by 1; if you drink a bottle of *double power potion*, your power doubles.

How long each battle lasts depends on how powerful you are. Each battle is described by six parameters: p_1, p_2, t_1, t_2, w_1, w_2. That means, if your power is less than p_1, you will be defeated; if your power is greater than p_2, you'll need t_2 seconds to defeat all the enemies. If your power is between p_1 and p_2 (inclusive), the time needed for the battle decreases linearly from t_1 to t_2. For example, if p_1=50, p_2=75, t_1=40, t_2=15, and your power is 55, then the battle lasts for 35 seconds. Note that the time needed for battles may be non-integers. The last two parameters, w_1 and w_2, represent the number of bottles of *stronger potion* and *double power potion* you got after winning the battle. Note that you don't have to drink these potions immediately. You can drink them later if that'll decrease the total time. You cannot drink potions during a battle, so the time needed for battles is not affected by the potions.

Given the list of battles, your task is to find a strategy to win all the battles as quickly as possible. Note that you must enter the battles in order. You cannot redo or skip any battle.

Input

There will be at most 25 test cases. Each test begins with two integers n and p ($1 \le n \le 1000$, $1 \le p \le 100$), the number of battles and your initial power. Each of the next n lines contains 6 integers p_1, p_2, t_1, t_2, w_1, w_2 ($1 \le p_1 < p_2 \le 100$, $1 \le t_2 < t_1 \le 100$, $0 \le w_1, w_2 \le 10$). The input is terminated by a test case with $n=p=0$, you should not process it.

Output

For each test case, print the shortest time (in seconds) to win all the battles, to two digits after the decimal point. If you cannot win all the battles, print "`Impossible`" (without quotes).

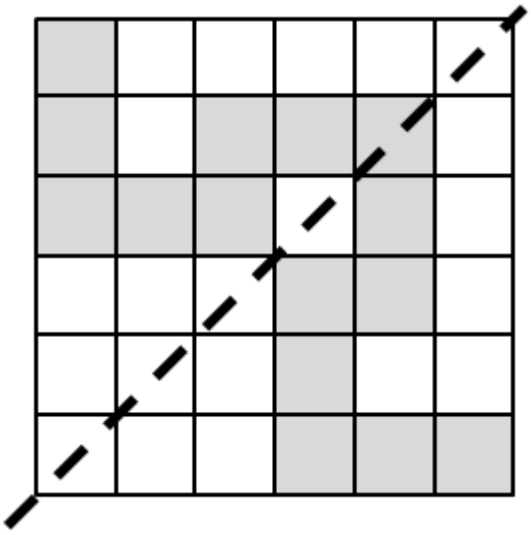
Sample Input	Output for Sample Input
1 55 50 75 40 15 10 0 2 55 50 75 40 15 10 0 50 75 40 15 10 0 3 1 1 2 2 1 0 5 1 2 2 1 1 0 1 100 100 1 0 0 1 7 4 15 35 23 0 0 1 1 2 3 2 1 0 0 0 0	35.00 60.00 41.00 31.73 Impossible

Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

G Optimal Symmetric Paths

Input: Standard Input
Output: Standard Output

You have a grid of n rows and n columns. Each of the unit squares contains a non-zero digit. You walk from the top-left square to the bottom-right square. Each step, you can move left, right, up or down to the adjacent square (you cannot move diagonally), but you cannot visit a square more than once. There is another interesting rule: your path must be symmetric about the line connecting the bottom-left square and top-right square. Below is a symmetric path in a 6x6 grid.



Your task is to find out, among all valid paths, how many of them have the minimal sum of digits?

Input

There will be at most 25 test cases. Each test case begins with an integer n ($2<=n<=100$). Each of the next n lines contains n non-zero digits (i.e. one of 1, 2, 3, …, 9). These n^2 integers are the digits in the grid. The input is terminated by a test case with n=0, you should not process it.

Output

For each test case, print the number of optimal symmetric paths, modulo 1,000,000,009.

Sample Input	Output for Sample Input
2 1 1 1 1 3 1 1 1 1 1 1 2 1 1 0	2 3

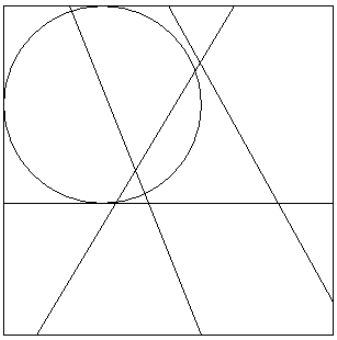
Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan

H Pieces and Discs

Input: Standard Input
Output: Standard Output

There is a rectangle on the Cartesian plane, whose bottom-left corner is (0,0), top-right corner is (L,W). You draw some line segments to divide the rectangle into pieces. Each line segment connects two points on the boundary of the original rectangle (these two points are guaranteed to be on different sides of the rectangle).

Finally, you draw some discs (a disc is a circle with its interior), and your task is to find out all the pieces each disc is intersecting with (i.e. pieces that have non-zero intersection area with the disc), and output their areas in increasing order. An example picture is shown below:



Input

There will be at most 100 test cases. Each test case begins with four integer n, m, L, W ($1<=n,m<=20$, $1<=L,W<=100$), where n is the number of line segments, m is the number of discs. Each of the next n lines describes a line segment with for integers x_1, y_1, x_2, y_2, that is a segment connecting (x_1,y_1) and (x_2,y_2). These two points are guaranteed to be on different sides of the original rectangle. Each of the last m line contains three integers x, y, R ($0<=x<=L$, $0<=y<=W$, $1<=R<=100$), indicating that the disc is centered at (x, y), whose radius is R. No two segments will be the same. Input is terminated by $n=m=L=W=0$.

Output

For each disc (same order as in input), print the number of pieces that the disc is intersecting with, and the areas of these pieces in a single line. The areas should be sorted in increasing order, and each area should be rounded to 2 digits after the decimal point. Print a blank line after each test case.

Sample Input

```
4 1 10 10
0 4 10 4
1 0 7 10
5 10 10 1
2 10 6 0
3 7 3
0 0 0 0
```

Output for Sample Input

```
4 0.50 10.03 10.77 18.70
```

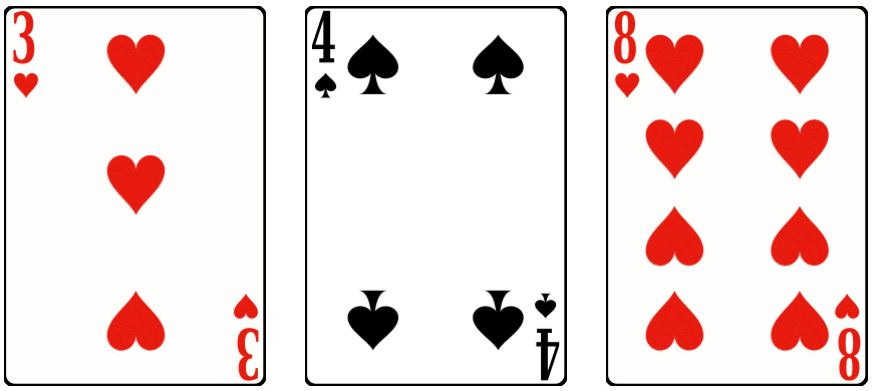
Problemsetter: Rujia Liu, Special Thanks: Jane Alam Jan

I Super Poker

Input: Standard Input
Output: Standard Output

I have a set of super poker cards, consisting of an infinite number of cards. For each positive integer p, there are exactly four cards whose value is p: Spade(S), Heart(H), Club(C) and Diamond(D). There are no cards of other values.

Given two positive integers n and k, how many ways can you pick up at most k cards whose values sum to n? For example, if n=15 and k=3, one way is 3H + 4S + 8H, shown below:



Input

There will be at most 20 test cases, each with two integers n and k ($1<=n<=10^9$, $1<=k<=10$). The input is terminated by n=k=0.

Output

For each test case, print the number of ways, modulo 1,000,000,009.

Sample Input	Output for Sample Input
2 1 2 2 2 3 50 5 0 0	4 10 10 1823966

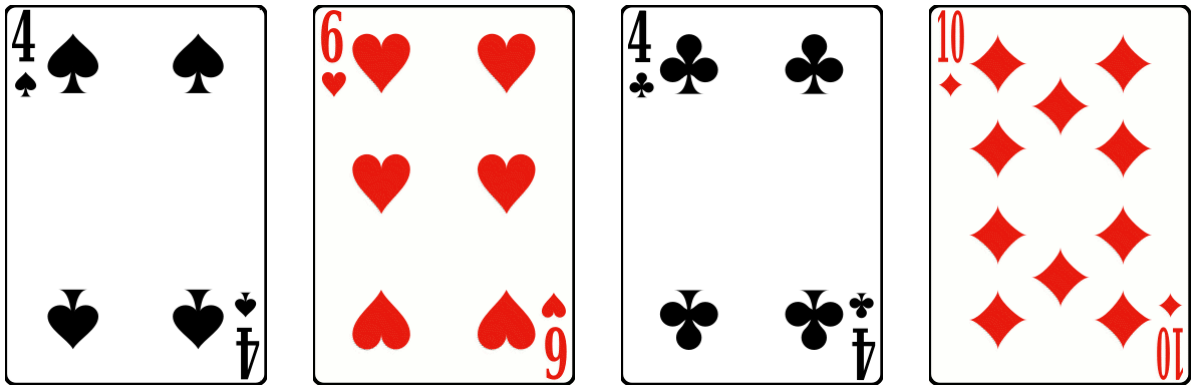
Problemsetter: Rujia Liu, Special Thanks: Jane Alam Jan

J Super Poker II

Input: Standard Input
Output: Standard Output

I have a set of super poker cards, consisting of an infinite number of cards. For each positive **composite** integer *p*, there are exactly four cards whose value is *p*: Spade(S), Heart(H), Club(C) and Diamond(D). There are no cards of other values. By "composite integer", we mean integers that have more than 2 divisors. For example, 6 is a composite integer, since it has 4 divisors: 1, 2, 3, 6; 7 is not a composite number, since 7 only has 2 divisors: 1 and 7. Note that 1 is not composite (it has only 1 divisor).

Given a positive integer *n*, how many ways can you pick up exactly one card from each suit (i.e. exactly one spade card, one heart card, one club card and one diamond card), so that the card values sum to *n*? For example, if *n*=24, one way is 4S+6H+4C+10D, shown below:



Unfortunately, some of the cards are lost, but this makes the problem more interesting. To further make the problem even more interesting (and challenging!), I'll give you two other positive integers *a* and *b*, and you need to find out all the answers for *n*=*a*, *n*=*a*+1, …, *n*=*b*.

Input

The input contains at most 25 test cases. Each test case begins with 3 integers *a*, *b* and *c*, where *c* is the number of lost cards. The next line contains *c* strings, representing the lost cards. Each card is formatted as *value*S, *value*H, *value*C or *value*D, where value is a composite integer. No two lost cards are the same. The input is terminated by *a*=*b*=*c*=0. There will be at most one test case where *a*=1, *b*=50,000 and *c*<=10,000. For other test cases, 1<=*a*<=*b*<=100, 0<=*c*<=10.

Output

For each test case, print *p* integers, one in each line. **Print a blank line after each test case.**

Sample Input	Output for Sample Input
12 20 2 4S 6H 0 0 0	0 0 0 0 0 0 1 0 3

Problemsetter: Rujia Liu, Special Thanks: Jane Alam Jan

K RMQ with Shifts

Input: Standard Input
Output: Standard Output

In the traditional RMQ (Range Minimum Query) problem, we have a static array A. Then for each query (L, R) (L<=R), we report the minimum value among A[L], A[L+1], …, A[R]. Note that the indices start from 1, i.e. the left-most element is A[1].

In this problem, the array A is no longer static: we need to support another operation shift(i_1, i_2, i_3, …, i_k) (i_1<i_2<...<i_k, k>1): we do a left "circular shift" of A[i_1], A[i_2], …, A[i_k].

For example, if A={6, 2, 4, 8, 5, 1, 4}, then shift(2, 4, 5, 7) yields {6, 8, 4, 5, 4, 1, 2}. After that, shift(1,2) yields {8, 6, 4, 5, 4, 1, 2}.

Input

There will be only one test case, beginning with two integers *n*, *q* (1<=*n*<=100,000, 1<=*q*<=250,000), the number of integers in array A, and the number of operations. The next line contains *n* positive integers not greater than 100,000, the initial elements in array A. Each of the next *q* lines contains an operation. Each operation is formatted as a string having no more than 30 characters, with no space characters inside. All operations are guaranteed to be valid. **Warning:** The dataset is large, better to use faster I/O methods.

Output

For each query, print the minimum value (rather than index) in the requested range.

Sample Input	Output for Sample Input
7 5 6 2 4 8 5 1 4 query(3,7) shift(2,4,5,7) query(1,4) shift(1,2) query(2,2)	1 4 6

Problemsetter: Rujia Liu, Special Thanks: Yiming Li & Jane Alam Jan