Problem A
Cornhusker
Time limit: 2 seconds

Corn farmers need to do pre-harvest yield estimates to determine the approximate number of bushels of corn their farm will produce. They do this to determine if they have enough storage space (grain bins) to store the harvested crop or if they'll have to store the corn elsewhere, like a co-op (which costs $$$). They also use these estimates when negotiating the future market prices of their corn. Estimates are typically done about a month or two before harvest. By this time, the ears have formed and the kernels on the ears are mostly developed (this makes counting the kernels easier).

According to the *University of Nebraska-Lincoln*, *Nebraska Extension for Educators*, the standard way to estimate corn yield is to calculate the number of bushels of corn per acre. To make the calculations easier, they use an area of 1/1000th of an acre, which, with 30" row spacing, is a section of one row about 17'5" long. Within that 17'5", five ears are chosen at random. For each ear, the number of kernels are counted by multiplying the number of rows of kernels around by the number of kernels over the length of the ear. The totals for each of the five ears are added together and then divided by five to determine the average number of kernels per ear of corn. This number is then multiplied by the total number of ears of corn in the 17'5" section of row. This gives you the total number of kernels in 1/1000th of an acre. This number is then divided by the *Kernel Weight Factor* (*KWF*). The *KWF* is a function of how wet (or dry) the growing season is and is typically a value between 75 (wet) and 95 (dry). The resulting quotient is the number of bushels/acre the farmer can expect to harvest.

For example, suppose that the average number of kernels per ear is 512 (16 kernels around by 32 kernels lengthwise), and there are 25 ears in the 17'5" of row with a *KWF* of 85. The farmer could then expect:

$$\frac{25 \times 512}{85} = 150\ bushels$$

Since farmers are quite conservative in their estimates, all calculations are done as integers with no rounding.

Input

Input consists of two lines. The first line contains 10 space separated integer values representing the number of kernels around (A) and number of kernels long (L) for each of five ears of corn ($8 \leq A \leq 24$), ($20 \leq L \leq 50$).

The second line contains 2 space separated integer values representing the number of ears of corn, N, in the 17'5" row ($10 \leq N \leq 50$) and the KWF ($75 \leq KWF \leq 95$).

Output

Output a single integer equal to the estimated number of bushels of corn per acre the farmer can expect given the input supplied.

Sample Input 1

```
16 32 16 32 16 32 16 32 16 32
25 85
```

Sample Output 1

```
150
```

Sample Input 2

```
14 30 15 32 15 34 14 34 16 32
27 75
```

Sample Output 2

```
172
```

Sample Input 3

```
16 24 16 34 16 40 14 30 16 35
28 95
```

Sample Output 3

```
150
```

Problem B
A Pivotal Question
Time limit: 3 seconds

Quicksort is a recursive sorting algorithm developed in 1959 by Tony Hoare. One of the major steps in the algorithm is the *partition* step: given an element p in the array (the *pivot* element) rearrange the elements in the array as shown below where all the values in X_L are $\leq p$ and all elements in X_R are $> p$.

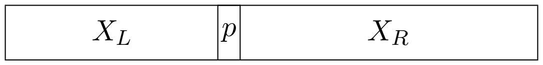


Figure B.1 below shows an array before and after it's been partitioned with the pivot element 13. Note that the elements in X_L and X_R are typically not in sorted order and either one of them could be empty.

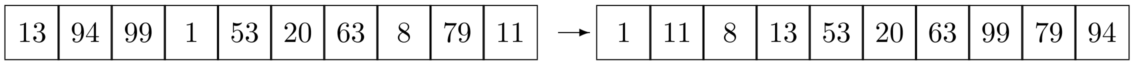


Figure B.1: An array before and after a partition

How a partition is executed and how a pivot element is selected are fascinating questions but are not of interest to us. What we would like you to do is the following: given an array, determine all the values that could be the pivot value assuming the array has been partitioned, or determine that the array has not been partitioned.

Input

Input starts with a positive integer n ($1 \leq n \leq 10^6$) denoting the size of the array. Following this are n positive integers indicating the values in the array. All values are unique and $\leq 10^6$.

Output

Output $m =$ the number of values in the array that could have served as pivot values to partition the array, followed by the pivot values in the order that they appear in the input. If $m > 100$ just output the first 100 of these pivot values. Note that a value of $m = 0$ indicates that the array is not partitioned.

Sample Input 1	Sample Output 1
`10 1 11 8 13 53 20 63 99 79 94`	`3 1 13 63`

This page is intentionally left blank.

Problem C
Prof. Fumblemore and the Collatz Conjecture
Time limit: 2 seconds

The ***Collatz function***, C(n), on positive integers is:

$$n/2 \text{ if } n \text{ is even and } 3n{+}1 \text{ if } n \text{ is odd}$$

The ***Collatz sequence***, CS(n), of a positive integer, n, is the sequence

$$\mathrm{CS}(n) = \mathrm{n}, \mathrm{C}(n), \mathrm{C}(\mathrm{C}(n)), \mathrm{C}(\mathrm{C}(\mathrm{C}(n))), \ldots$$

For example, CS(12) = 12, 6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, . . .

The ***Collatz Conjecture*** (also known as the *3n+1 problem*) is that CS(n) for every positive integer n eventually ends repeating the sequence 4, 2, 1. To date, the status of this conjecture is still unknown. No proof has been given and no counterexample has been found up to very large values.

Prof. Fumblemore wants to study the problem using *Collatz sequence types*. The *Collatz sequence type* (CST) of an integer n, CST(n) is a sequence of letters E and O (for even and odd) which describe the parity of the values in CS(n) up to but not including the first power of 2. So,

$$\mathrm{CST}(12) = \mathrm{EEOEO}$$

Note that

$$\mathrm{CS}(908) = 908, 454, 227, 682, 341, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 3, 2, \ldots$$

so 12 and 908 have the same CST.

Prof. Fumblemore needs a program which allows him to enter a sequence of E's and O's and returns the **smallest** integer n for which the given sequence is CST(n).

Notes:

- E's are even numbers which are not powers of 2,
- O's are odd numbers greater than 1.
- The last letter in a sequence must be an O (if C(n) is a power of 2, so is n)
- There can not be two O's in succession (C(odd) = even)
- Since, Prof. Fumblemore does not type well, you must check that the input sequence is valid before attempting to find n. That is, the sequence contains only E's and O's, ends in O and no two O's are adjacent.

Input

Input consists of one line containing a string of up to 50 letters composed of E's and O's.

Output

There is one line of output that consists of the string `INVALID` if the input line is invalid, or a single decimal integer, n, such that n is the *smallest* integer for which CST(n) is the input sequence. Input will be chosen such that $n \leq 2^{47}$.

Sample Input 1

```
EEOEO
```

Sample Output 1

```
12
```

Sample Input 2

```
EEOOEO
```

Sample Output 2

```
INVALID
```

Problem D
Double Up
Time limit: 7 seconds

A Double Up game consists of a sequence of n numbers $a_1, \dots, a_n$, where each a_i is a power of two. In one move one can either remove one of the numbers, or merge two identical adjacent numbers into a single number of twice the value. For example, for sequence $4, 2, 2, 1, 8$, we can merge the 2s and obtain $4, 4, 1, 8$, then merge the 4s and obtain $8, 1, 8$, then remove the 1, and, finally, merge the 8s, obtaining a single final number, 16. We play the game until a single number remains. What is the largest number we can obtain?

Input

The input consists of two lines. The first line contains n ($1 \le n \le 1000$). The second line contains numbers $a_1, \dots, a_n$, where $1 \le a_i \le 2^{100}$ for each i.

Output

The ouput consists of a single line containing the largest number that can be obtained from the input sequence $a_1, \dots, a_n$.

Sample Input 1	Sample Output 1
5 4 2 2 1 8	16

Problem E
Three Spheres and a Tetrahedron
Time limit: 2 seconds

Given a tetrahedron $OABC$ with vertices O, A, B and C.

There is a sphere, $S1$ (red, center $Q1$), inscribed in the tetrahedron tangent to the inside of each face OAB (gray), OAC (brown), OBC (magenta) and ABC (cyan and black).

There is a second sphere, $S2$ (green, center $Q2$), tangent to the (extended) inside of OAB, OAC and OBC and to the outside of ABC. (There is actually such a sphere for each face, tangent to the outside of the face and the inside of the other extended faces).

There is a third larger sphere, $S3$ (blue, center $Q3$), which passes thru vertices A, B and C and is tangent to each of $S1$ and $S2$ so the outside of the smaller spheres is tangent to the inside of the largest sphere (see Figure 1, below, for two different views. Triangle ABC is cyan in the first picture and black in the second one for clarity):

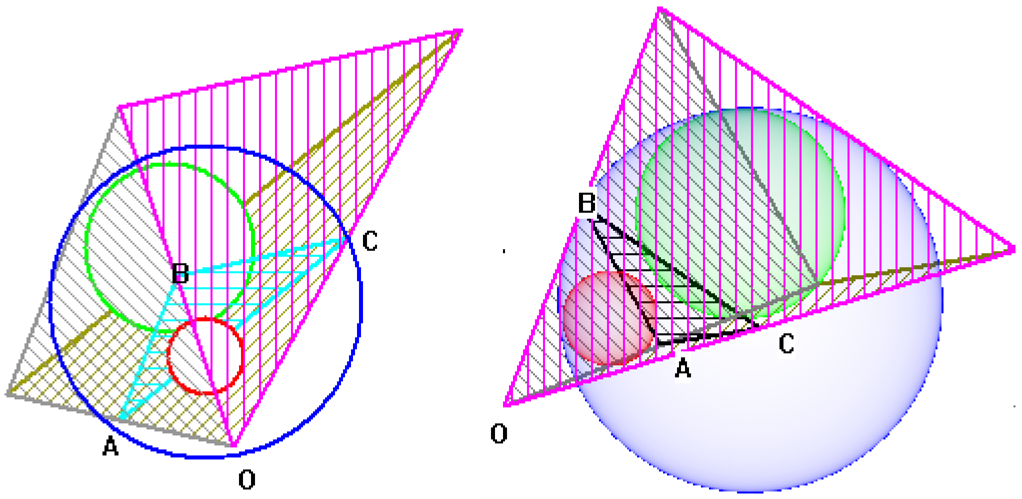


Figure 1

The following figures give several views of the tetrahedron and spheres.

Figure 2 shows the view along OA, which shows the two smaller spheres tangent to OAB and OAC (left). The view along BC shows the two smaller spheres tangent to OBC and tangent on opposite sides of ABC (right):

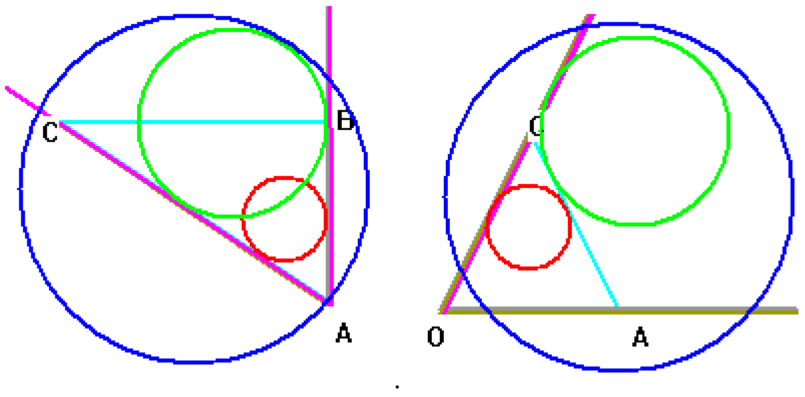


Figure 2

Figure 3 shows $S3$ passing through A, B and C and tangent to $S1$ and $S2$. On the left, the view perpendicular to the plane of triangle A,B,$Q3$ shows $S3$ passing through A and B. In the center, the view perpendicular to the plane of triangle A,C,$Q3$ shows $S3$ passing through A and C. On the right, the view perpendicular to the plane of triangle $Q1, Q2, Q3$ (the centers of the three spheres) shows $S1$ and $S2$ tangent to the inside of $S3$.

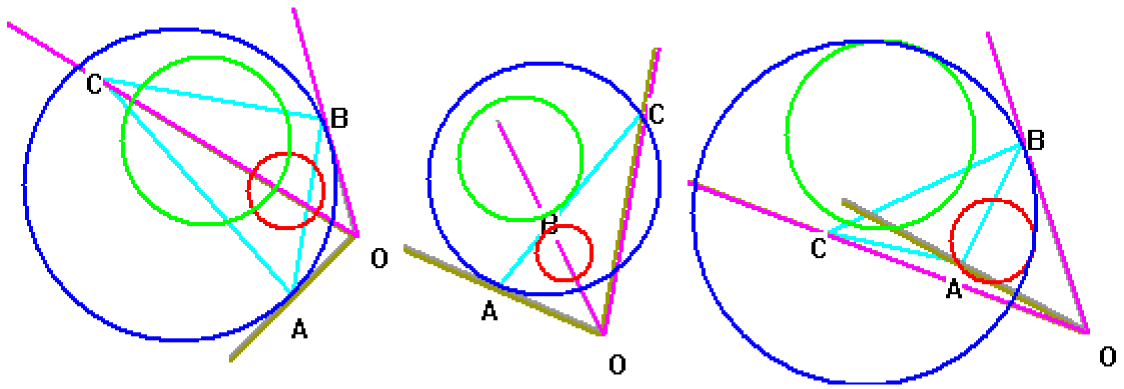


Figure 3

Write a program which takes as input the vertices O, A, B and C and computes the center and radius of the big sphere (which entails finding the other two spheres).

O will be the origin (0,0,0). A will lie on the positive x-axis (Ax,0,0), B will be on the $xy-$plane (Bx,By,0) and C will be in the first orthant (Cx,Cy,Cz). Ax, By and Cz will be strictly positive and the remaining values will be non-negative.

Input

The input consists of a single line containing six double precision decimal values Ax, Bx, By, Cx, Cy and Cz in that order (as described above), $(0 < Ax, By, Cz \leq 10)$ and $(0 \leq Bx, Cx, Cy \leq 10)$.

Output

The single line of output contains four decimal values to four decimal places: $center_x$, $center_y$, $center_z$ and $radius$ of the big sphere.

Sample Input 1	Sample Output 1
`2 3 2 3 1 4`	`2.8563 0.8218 1.8305 2.1816`

Sample Input 2	Sample Output 2
`1 0 2 0 0 3`	`1.0000 1.2500 1.6667 2.0833`

Problem F
Forest for the Trees
Time limit: 5 seconds

You have sent a robot out into the forest, and it has gotten lost. It has a sensor that will detect all the trees around itself regardless of any occlusions, but unfortunately in this forest, all trees look alike. You do have a map of all trees in the forest, represented as (x, y) points. Conveniently, since this used to be a tree farm, all trees are at integer coordinates, though not all coordinates are occupied. The robot's sensor tells you the x and y distance to each tree within range, relative to the front of the robot. However, the robot is heading in an unknown direction relative to the map, so each sensor reading is given as a tuple of (distance to the right of the robot, distance forward of the robot) and either value can be negative since the robot can sense in all directions. Helpfully, the robot will always place itself at integer coordinates and aligned to the positive or negative x or y axis, and will never be at the same location as a tree. Can you find out where the robot is?

Input

The first line of input contains three integers: n_t, the number of trees in the forest, n_s, the number of trees sensed by the robot, and r_{max}, the maximum Manhattan distance (sum of x and y distances) of any sensor reading. The next n_t lines each contain two integers representing the (x, y) locations of all the trees relative to a global coordinate system. The final n_s lines each contain two integers. The first integer in the i^{th} sensor reading, $s_{i,x}$, represents the distance to the tree along the axis perpendicular to the robot's heading and the second integer $s_{i,y}$ represents the distance along the axis parallel to the robot's heading. You can assume that $|s_{i,x}| + |s_{i,y}| \leq r_{max}$ for all i. You may also assume $0 < n_t \leq 5000$, $0 < n_s \leq 1000$, $0 < r_{max} \leq 1000$, and all tree locations have x and y coordinates $-100,000 \leq x, y \leq 100,000$.

Output

Print one of the following: the x, y location of the robot, printed as two integers separated by a space; "Impossible" if there is no location in the map that could produce the given sensor values, or "Ambiguous" if two or more distinct locations and/or orientations could produce the given sensor values.

Sample Input 1	Sample Output 1

4 4 100
1 1
2 2
2 1
3 3
0 1
0 2
-1 2
-2 3
	0 1

This page is intentionally left blank.

Problem G
Pearls
Time limit: 5 seconds

Nikoli's Jewelry Store in Puzzletown sells a line of necklaces consisting of black and white pearls. The pearls in the necklace are firmly glued to a cord of length k, where each unit of cord length either holds a pearl or is empty. Each necklace is displayed on a rectangular velvet-lined surface divided into a grid, where each cell of the grid either holds a pearl, or contains a unit of empty cord, or is unoccupied by either pearl or cord. All cord sections are either horizontal or vertical. A properly-displayed necklace corresponds to a closed, non-self-intersecting path connecting some of the cells of the display.

Because this is, after all, Puzzletown, Nikoli uses some tricky rules governing how the necklace is to be displayed, namely, the rules of a puzzle called "Masyu." When the the necklace is set down along the path (the spacing units on the string match the spacing of the cells on the display surface), the pearls satisfy the constraints of the Masyu puzzle, i.e.,

- A white pearl may not be set down on a cell containing a path corner; in addition, at least one of the two adjacent cells that extend the path through the pearl must contain a corner.
- A black pearl must be placed in a cell containing a path corner; in addition, neither of the two cells extending the path through the black pearl may contain a corner.

An example of a necklace correctly displayed is shown in Figure G.1 (this also corresponds to sample input 1).

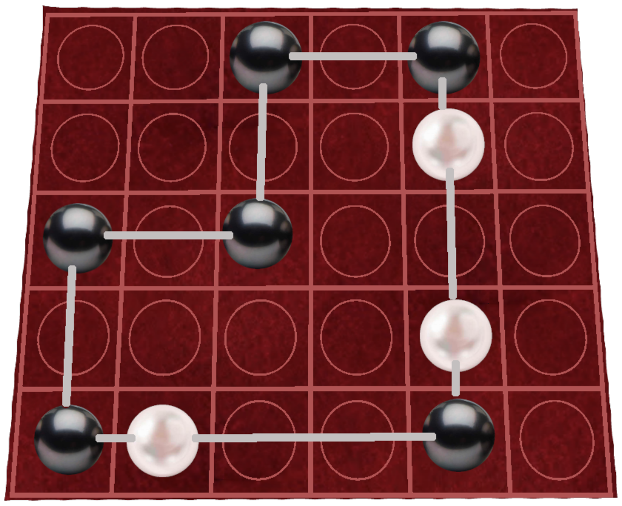


Figure G.1: A necklace of length 16 on its display platform

Nikoli's clientele are somewhat picky, so he places three further restrictions on his necklaces. At least half of the necklace's length consists of pearls rather than empty sections of cord. And because black pearls are more desirable (or at least, more expensive) than white ones, the wealthy residents of Puzzletown insist that there be at least twice as many black pearls as white ones. Finally, no two pearls are ever separated by a gap of empty cord longer than five units.

Nikoli sometimes finds that once he has created a necklace according to these restrictions, he is not able to display it according to the rules above. Please help him!

Input

The first line contains 3 integers k, n, and m, where k ($5 \le k \le 60$) is the length of the cord and n and m ($5 \le n, m \le 50$) are respectively the number of rows and columns of the velvet grid. The upper-left cell is row 1, column 1. The second line contains a string of length k consisting only of the characters '`B`', '`W`', and '`.`' (for black pearl, white pearl, and empty cord segment). The first character will always be a pearl—either `B` or `W`. The third line contains two integers r and c ($1 \le r \le n$, $1 \le c \le m$), the row and column of the grid that contains the first pearl in the string.

Output

If there exists a proper way to display the necklace within the given grid boundaries, print a path description of the necklace layout, assuming the first pearl in the string is located at row r, column c of the grid and the path describes the pearls and empty spaces in the same sequence as the input string. The path description should consist of the letters `N`,`S`,`E`, and `W`, indicating whether the path proceeds north, south, east, or west from the current cell. The path should be closed and should not intersect itself. If there is more than one such path, output the one whose description is alphabetically the smallest.

If there is no possible path satisfying the Masyu constraints, output `impossible`

Sample Input 1

```
16 5 6
B.B.B.BW.WB..WB.
3 1
```

Sample Output 1

```
EENNEESSSSWWWWNN
```

Sample Input 2

```
6 5 5
W..B.B
3 3
```

Sample Output 2

```
impossible
```

Problem H
A (Fast) Walk in the Woods
Time limit: 3 seconds

Brice Bilson loves to take jogs in a nearby forest known as Orthogonal Woods. The forest gets that name as the paths – all two-way – are laid out along an orthogonal grid, with all turns being 90 degrees. Brice is a bit persnickety when it comes to his jogs, and always follows a set of rules when he reaches an intersection of two or more paths. These rules are

1. If there are three remaining branches, Brice takes the middle one.
2. If there are just two remaining branches, Brice takes the one on his left.
3. If there are no branches to take, Brice ends his jog and walks to the nearest exit.

Brice is persnickety in another way too. He has assigned each path an "interest value", which is a positive integer indicating how interesting that path is to jog. The higher the value, the more interesting the path is. If the value of a path is n, then Brice will jog on that path no more than n times in his jog. After the n^{th} pass that path will cease to exist as far as Brice is concerned (so, for example, any three-branch intersection using that path now becomes a two-branch intersection and any two-branch intersection becomes a one-branch intersection). An example is shown in Figure H.1 below:

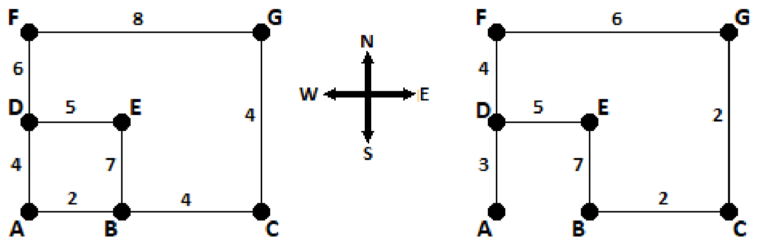


Figure H.1: Sample park corresponding to Sample Input 1

Suppose Brice enters the park at intersection D heading north in the figure on the left, where the numbers next to each path indicate his interest levels. His travels takes him on the route DFGCBADFGCBA at which point we reach the figure on the right, showing the updated interest levels of each path and the "removal" of the path from A to B since it's now been traversed 2 times. From intersection A Brice now traverses the route ADFGCBEDA at which point he hits a dead end and ends his jog.

Input

Input starts with two integers n and m ($2 \leq n \leq 2\,500$) giving the number of intersections and the number of paths between intersections. The next line contains n pairs of integers giving the locations of the intersections. Intersections are numbered from 1 to n in the order they are presented and all location values x, y satisfy $0 \leq x, y \leq 10^6$. After this are m lines each containing three integers i j k ($1 \leq i, j \leq n, 1 \leq k \leq 10^6$)

indicating that a path exists between intersections i and j with interest level k. All paths will be either vertical or horizontal and will not touch any other vertices other than the specified intersection points. The final line of input contains an integer s ($1 \leq s \leq n$) and a character $d \in \{$N,S,E,W$\}$ indicating that Brice starts his jog by taking the path in direction d from intersection s. There will always be a path heading in direction d from vertex s.

Output

Output the location where Brice ends his jog.

Sample Input 1

```
7 8
0 0 5 0 12 0 0 5 5 5 0 10 12 10
1 2 2
2 3 4
4 5 5
6 7 8
1 4 4
2 5 7
3 7 4
4 6 6
4 N
```

Sample Output 1

```
0 0
```

Problem I
ISBN Conversion
Time limit: 2 seconds

ISBN 3-540-42580-2

9 783540 425809

Barcode with ISBN-10 above and ISBN-13 below

An ISBN (International Standard Book Number) is a unique identifier assigned to a distinct edition/version of a published book (for example, hardcover and paperback versions of the same edition get different ISBNs). ISBNs assigned before 2007 were 10 digits long (the ISBN-10 standard), and ISBNs assigned on or after January 1, 2007, are 13 digits long (the ISBN-13 standard). Some books issued before 2007 that are still in print have both an original ISBN-10 and a matching ISBN-13, and some newer books are also given both an ISBN-10 and an ISBN-13, the former for backward-compatibility purposes. That "double identity" situation is the basis for this problem, which requires you to convert valid ISBN-10s to their corresponding ISBN-13s.

The last digit of any ISBN is a *checksum digit* that can be used for simple error detection. ISBN-10 and ISBN-13 use different rules for computing/verifying this last digit:

1. ISBN-10: If the 10 digits from left to right are $d_1, d_2, \ldots, d_{10}$ (so d_{10} is the checksum digit), and if

   $$S = 10 \cdot d_1 + 9 \cdot d_2 + 8 \cdot d_3 + \ldots + 2 \cdot d_9 + 1 \cdot d_{10}$$

   (coefficients decrease from 10 to 1), then S must be a multiple of 11, i.e., $S \equiv 0 \pmod{11}$.

2. ISBN-13: If the 13 digits from left to right are $d_1, d_2, \ldots, d_{13}$ (so d_{13} is the checksum digit), and if

   $$S = 1 \cdot d_1 + 3 \cdot d_2 + 1 \cdot d_3 + 3 \cdot d_4 + \ldots + 3 \cdot d_{12} + 1 \cdot d_{13}$$

   (odd-indexed digits are multiplied by 1, even-indexed digits are multiplied by 3), then S must be a multiple of 10, i.e., $S \equiv 0 \pmod{10}$.

It is not hard to see that each rule uniquely determines the checksum digit (given the other digits).

X Factor: Note the following small but important detail for ISBN-10 that does not apply to ISBN-13: because the modulus is 11, the value of the checksum digit lies in $\{0, 1, 2, \ldots, 9, 10\}$, and in the special case that the value of the checksum digit is 10, it is written as X so that only one character is required. So, for example, 039428013X is a valid ISBN-10.

Hyphens: Technically an ISBN-10 consists of four parts, one of which is the checksum digit. (The exact rules defining the first three parts are complicated, so we will not deal with them here.) Two adjacent parts can optionally be separated by a hyphen, which means that an ISBN-10 may contain up to three hyphens, but it cannot begin or end with a hyphen, and it cannot contain consecutive hyphens. If there are three hyphens, one must separate the checksum digit from the digit that precedes it (if there are fewer than three hyphens, there may or may not be a hyphen between the checksum digit and the digit that precedes it). So, for the purposes of this problem, the following are valid ISBN-10s:

039428013-X
0-39-428013X
3-540-42580-2
3540425802

And the following are *invalid* ISBN-10s (the first two because of a hyphen-placement error, the last because it fails the checksum test above):

3-540-4258-02
3-540-425802-
0-14-028333-3

How do you convert an ISBN-10 to an ISBN-13? Simply (i) prepend the three digits 978, (ii) remove the old checksum digit, and (iii) append a new checksum digit as determined by the ISBN-13 rule.[1] To keep things simple, maintain the positions of any existing hyphens, and follow the prepended 978 with a hyphen.

Input

The first line of input contains an integer, T $(1 \leq T \leq 25)$, the number of (possibly invalid) ISBN-10s to process. This is followed by T lines, each of which contains a nonempty string of length between 10 and 13, inclusive. Each character is either a base-10 digit (`0` to `9`), a hyphen (`-`), or `X` (capital X).

Output

For each test case, if the candidate ISBN-10 is not valid according to the details given above, output a line containing "`invalid`". Otherwise, output a line containing the corresponding ISBN-13.

Sample Input 1

```
4
3-540-4258-02
039428013X
3-540-42580-2
0-14-028333-3
```

Sample Output 1

```
invalid
978-0394280134
978-3-540-42580-9
invalid
```

[1] In general, an ISBN-13 can begin with three digits other than 978, but only 978 can be prepended to an ISBN-10 to form the matching ISBN-13.

Problem J
Base Hi-Lo Game
Time limit: 2 seconds

For this problem, you will write a program that plays a guessing game to figure out a number of up to 64 digits in a specified base (2–62). Digits are: `0-9, A-Z, a-z` in that order.

This is an interactive problem. All lines sent by your program must be **newline terminated and flushed out** (you may have to disable buffering). All responses received by your program will be **newline terminated lines**.

Your program will make a guess of what the number might be, and the judging system will respond with a string giving you information about each digit in your guess.

There is a limit on the number of guesses you are allowed to make for each test case, after which point, if you have not guessed the number, your program will be terminated and given a `WA` (wrong answer) judgment. The number of guesses you are allowed is $floor(log_2(base)) + 2$. To make it interesting, it's possible that the judging system may try to *cheat* you and give you faulty information back. If the judging system does decide to cheat, it will cheat only once, for a single digit.

If the number you guess is correct, the judging system will *always* respond with the string "`correct`" (with no quotes).

Sample Interaction

When your program starts, it should read two space separated decimal integer values: the base B of all the test cases, $(2 \leq B \leq 62)$, and the number of test cases, N $(1 \leq N \leq 100)$.

Your program will then read a decimal value, which is the number of digits D in the first test case, $(1 \leq D \leq 64)$. Your program will then respond with a line consisting of a D digit, base B value which is your first guess. You will then read back a line containing a string from the judging system. The action your program takes now depends on the string you read from the judging system.

- Case 1: If the value of the string is "`correct`", you have successfully guessed the number. If there are more test cases, your program should go back and wait for the length D of the next test case. If there are no more test cases, your program should exit.
- Case 2: The string is a D character string consisting of characters in the set: `[+,-,=]`. Each character is an indication as to the accuracy of the digit in the corresponding position of your guess. A + means the digit in that position is too small. A − means the digit in that position is too big. An = means the digit is correct. You will then make a new guess based on the information you got back. Your program will repeat this process until you receive a "`correct`" response, or the judge terminates your program because it took too many guesses.

If at any point during the guessing dialog you detect that the judging system is cheating, you should send back a line with the word "`cheater`" on it (without quotes). If the judging system was, in fact, cheating, you will receive a "`correct`" (without quotes) response back. If the judging system was not cheating, your program will be terminated and it will receive a `WA` response. An example of cheating might be if the judging system responds in different ways to the same guess. Another example might be contradictory information: a judging system response to an earlier guess conflicts with the response of a later guess. The judging system may cheat only once per test case.

For **Sample Interaction 1**, the correct value is `GNY23`. For **Sample Interaction 2**, the correct values are `555` and `9`.

Sample Interaction 1

Read	Write
36 1 5	
	00000
+++++	
	55555
+++--	
	AAA33
+++-=	
	GGG13
=+++=	
	GNN23
==+==	
	GNT23
(no response - too many guesses - WA)	

Sample Interaction 2

Read	Write
10 2 3	
	543
=++	
	554
==+	
	555
correct 1	
	2
+	
	5
+	
	7
+	

```
8
```

```
(no response - too many guesses - WA)
```

Problem K
Plus Minus Four Squares
Time limit: 1 second

Every non-negative integer *n* may be written as the sum of the squares of four integers:

$$n = a^2 + b^2 + c^2 + d^2$$

By allowing subtraction, *n* may be written in many more ways; in fact infinitely many.

In this problem you will count the number of different ways to express an input *n* as a sum or difference of four squares with several restrictions:

First, we need to decide what *different* means.

Any of a, b, c, d may be replaced by its negative. We do not want to count these as *different* so we will only count different *squared* values.

Reordering a, b, c, d does not give a *different* representation.

So, we define a *plus minus four square* representation of a non-negative integer *n* as a sequence of four perfect squares in *non-increasing* order with plus or minus signs whose computation results in *n*.

In addition, we add the following restrictions:

- The first square must be no more than *n* to avoid having infinitely many representations.
- If the same square appears multiple times **all** appearances must be preceded by (a possibly implicit) plus sign or **all** must be preceded by a minus sign. This avoids something like:

  ```
  64 + 36 - 36 + 0
  ```

- A square of zero *must* be preceded by a plus sign.

For example, the only sums of squares which add to 64 are:

```
64 + 0 + 0 + 0
16 + 16 + 16 + 16
```

If we allow minus signs with the above additional restrictions we have the following which each sum up to 64:

```
64 + 25 - 16 - 9
64 - 25 + 16 + 9
64 + 0 + 0 + 0
49 + 49 - 25 - 9
49 + 36 - 25 + 4
49 + 25 - 9 - 1
49 + 16 - 1 + 0
```

```
36 + 36 - 9 + 1
36 + 36 - 4 - 4
36 + 25 + 4 - 1
36 + 16 + 16 - 4
16 + 16 + 16 + 16
```

Write a program which takes as input a *non-negative* integer n and outputs a count of the number of different *four square plus minus* representations of n.

Input

Input consists of one line containing a single non-negative decimal integer ($0 < n \leq 5000$).

Output

There is one line of output that consists of a single decimal integer giving a count of the number of different *four square plus minus* representations of n.

Sample Input 1	Sample Output 1
64	12

Sample Input 2	Sample Output 2
65	10

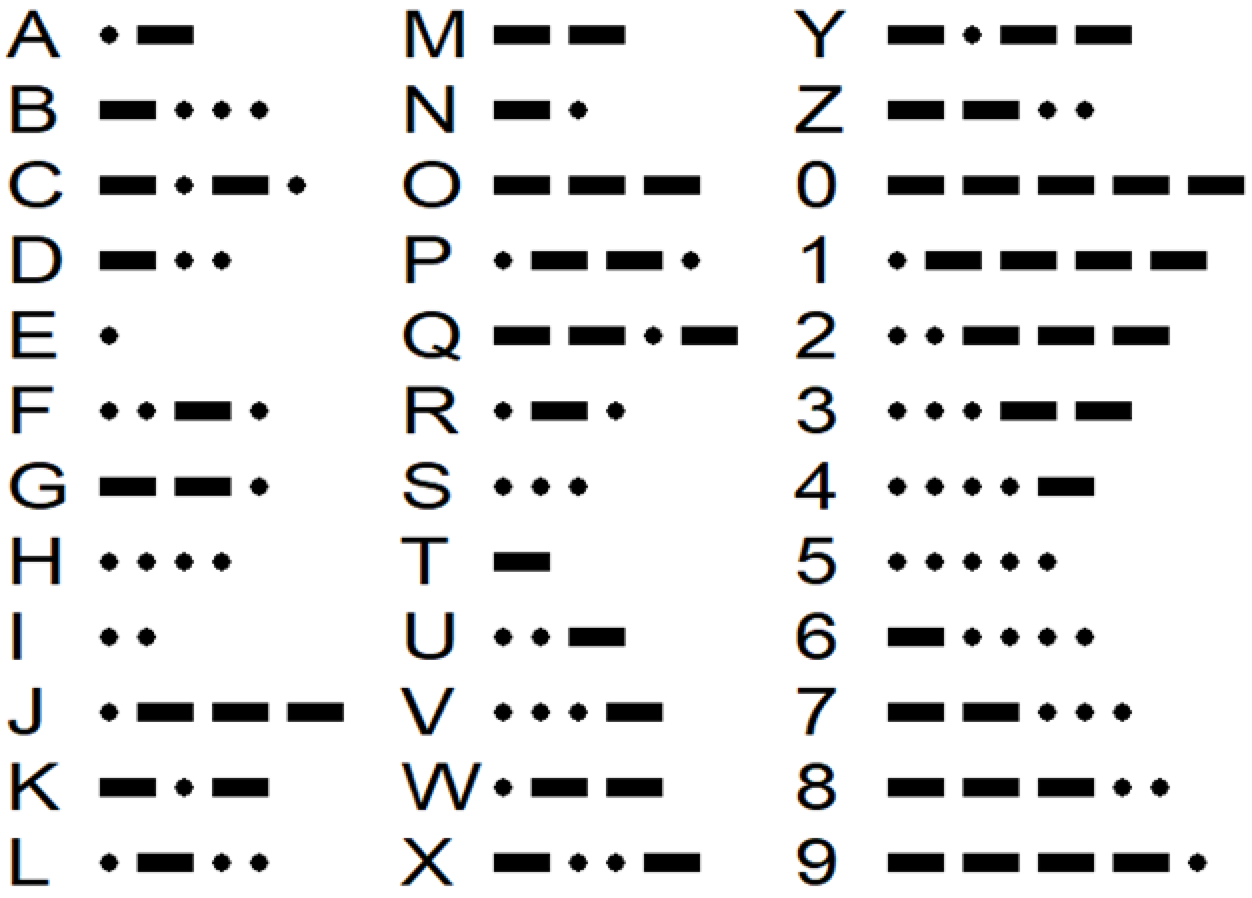
Sample Input 3	Sample Output 3
2023	245

Problem L
Make Your Own Morse Code Palindrome
Time limit: 3 seconds

A *palindrome* is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as *madam* or *racecar* or *1881*. *Phrase palindromes* ignore capitalization, punctuation, and word boundaries. For example:

Madam I'm Adam.

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs. Morse code is named after Samuel Morse, one of the inventors of the telegraph. The International Morse code for letters and digits (the code does not distinguish between upper and lower case) is:



A	• ▬	M	▬ ▬	Y	▬ • ▬ ▬
B	▬ • • •	N	▬ •	Z	▬ ▬ • •
C	▬ • ▬ •	O	▬ ▬ ▬	0	▬ ▬ ▬ ▬ ▬
D	▬ • •	P	• ▬ ▬ •	1	• ▬ ▬ ▬ ▬
E	•	Q	▬ ▬ • ▬	2	• • ▬ ▬ ▬
F	• • ▬ •	R	• ▬ •	3	• • • ▬ ▬
G	▬ ▬ •	S	• • •	4	• • • • ▬
H	• • • •	T	▬	5	• • • • •
I	• •	U	• • ▬	6	▬ • • • •
J	• ▬ ▬ ▬	V	• • • ▬	7	▬ ▬ • • •
K	▬ • ▬	W	• ▬ ▬	8	▬ ▬ ▬ • •
L	• ▬ • •	X	▬ • • ▬	9	▬ ▬ ▬ ▬ •

A word, number or phrase is a *Morse Code Palindrome* if the morse code for the letters and digits in the word, number or phrase reads the same backwards or forwards (ignoring spaces between character codes.

For example:

159
● ▬ ▬ ▬ ▬   ● ● ● ● ●   ▬ ▬ ▬ ▬ ●

Footstool
● ● ▬ ●   ▬ ▬ ▬   ▬ ▬ ▬   ▬   ● ● ●   ▬   ▬ ▬ ▬   ▬ ▬ ▬   ● ▬ ● ●

Write a program which takes as input a string and determines the *smallest* number of characters to append to the end of the word to make it a Morse Code Palindrome.

Input

Input consists of a single line containing a string of up to 30 capital letters and/or digits possibly including spaces and punctuation.

Output

The output consists of a single line.

If the input string is already a Morse Code Palindrome, output the digit 0. Otherwise output the number of characters to append, followed by a single space followed by a string containing capital letters and/or digits to append to make the input a Morse Code Palindrome.

Since there may be more than one valid answer (see samples 2 and 3 below), a result will be judged correct if the number of appended characters is less than or equal to the judges' answer and the input followed by your output is a Morse Code Palindrome.

Sample Input 1	Sample Output 1
`FOOT`	`1 L`

Sample Input 2	Sample Output 2
`FOOTS`	`3 0QI`

Sample Input 3	Sample Output 3
`FOOTS`	`3 0GD`

Sample Input 4	Sample Output 4
`FOOTSTOOL`	`0`