Problem A - Complete the sequence

You probably know those quizzes in Sunday magazines: given the sequence 1, 2, 3, 4, 5, what is the next number? Sometimes it is very easy to answer, sometimes it could be pretty hard. Because these “sequence problems” are very popular.

Notice that some of the quizzes can be solved by describing the sequence by polynomials. For example, the sequence 1, 2, 3, 4, 5 can be easily understood as a trivial polynomial. The next number is 6. But even more complex sequences, like 1, 2, 4, 7, 11, can be described by a polynomial. In this case, $1/2 \times n^2 - 1/2 \times n + 1$ can be used. Note that even if the members of the sequence are integers, polynomial coefficients may be any real numbers.

Input
There is a single positive integer N (<120) on the first line of input. It stands for the number of test cases to follow. Each test case consists of two lines. First line of each test case contains two integer numbers S and C separated by a single space, $1 \le S < 100$, $1 \le C < 100$, $S + C \le 100$. The first number, S, stands for the length of the given sequence, the second number, C is the amount of numbers you are to find to complete the sequence.

The second line of each test case contains S integers X_1, X_2, ..., X_S separated by a space. These numbers form the given sequence. The sequence can always be described by a polynomial $P(n)$ such that for every i, $X_i = P(i)$. Among these polynomials, you should find the polynomial with the lowest possible degree. This polynomial should be used for completing the sequence.

Output
For every test case, your program must print a single line containing C integers, separated by a space. These numbers are the values completing the sequence according to the polynomial of the lowest possible degree. In other words, you should print values $P(S+1)$, $P(S+2)$, ..., $P(S+C)$. It is guaranteed that all the integers of the input and output are non-negative and will fit into the standard integer type.

Sample input
```
4
6 3
1 2 3 4 5 6
8 2
1 2 4 7 11 16 22 29
10 2
1 1 1 1 1 1 1 1 1 2
1 10
3
```

Sample output
```
7 8 9
37 46
11 56
3 3 3 3 3 3 3 3 3 3
```

Problem B - Transmitters

In a wireless network with multiple transmitters sending on the same frequencies, it is often a requirement that signals don't overlap, or at least that they don't conflict. One way of accomplishing this is to restrict a transmitter's coverage area. This problem uses a shielded transmitter that only broadcasts in a semicircle.

A transmitter *T* is located somewhere on a 1,000 square meter grid. It broadcasts in a semicircular area of radius *r*. The transmitter may be rotated any amount, but not moved. Given *N* points anywhere on the grid, compute the maximum number of points that can be simultaneously reached by the transmitter's signal. Figure 1 shows the same data points with two different transmitter rotations.

Figure 1a

Figure 1b

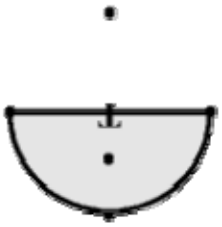


Figure 2

All input coordinates are integers (0-1000). The radius is a positive real number greater than 0. Points on the boundary of a semicircle are considered within that semicircle. There are at most 150 unique points to examine per transmitter. No points are at the same location as the transmitter.

Input
Input consists of information for one or more independent transmitter problems. Each problem begins with one line containing the (*x*, *y*) coordinates of the transmitter followed by the broadcast radius, *r*. The next line contains the number of points *N* on the grid, followed by *N* sets of (*x*, *y*) coordinates, one set per line. The end of the input is signaled by a line with a negative radius. Figures 1 and 2 represent the data in the first two example data sets below, though they are on different scales. Figures 1a and 2 show transmitter rotations that result in maximal coverage. There will be at most 50 test cases.

Output
For each transmitter, the output contains a single line with the maximum number of points that can be contained in some semicircle.

Sample input
```
25 25 3.5
7
25 28
23 27
27 27
24 23
26 23
```

```
24 29
26 29
350 200 2.0
5
350 202
350 199
350 198
348 200
352 200
995 995 10.0
4
1000 1000
999 998
990 992
1000 999
100 100 -2.5
```

Sample output

```
3
4
4
```

Problem C –Sorting It All Out

An ascending sorted sequence of distinct values is one in which some form of a less-than operator is used to order the elements from smallest to largest. For example, the sorted sequence A, B, C, D implies that A < B, B < C and C < D. In this problem, we will give you a set of relations of the form A < B and ask you to determine whether a sorted order has been specified or not.

Input
Input consists of multiple problem instances. Each instance starts with a line containing two positive integers *n* and *m*. The first value indicates the number of objects to sort, where $2 \leq n \leq 26$. The objects to be sorted will be the first *n* characters of the uppercase alphabet. The second value *m* indicates the number of relations of the form A<B which will be given in this problem instance. Next will be *m* lines, each containing one such relation consisting of three characters: an uppercase letter, the character "<" and a second uppercase letter. No letter will be outside the range of the first *n* letters of the alphabet. Values of $n = m = 0$ indicate end of input. There will be at most 110 test cases.

Output
For each problem instance, output consists of one line. This line should be one of the following three:

```
Sorted sequence determined after xxx relations: yyy...y.
Sorted sequence cannot be determined.
Inconsistency found after xxx relations.
```

where xxx is the number of relations processed at the time either a sorted sequence is determined or an inconsistency is found, whichever comes first, and yyy...y is the sorted, ascending sequence.

Sample Input
```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
0 0
```

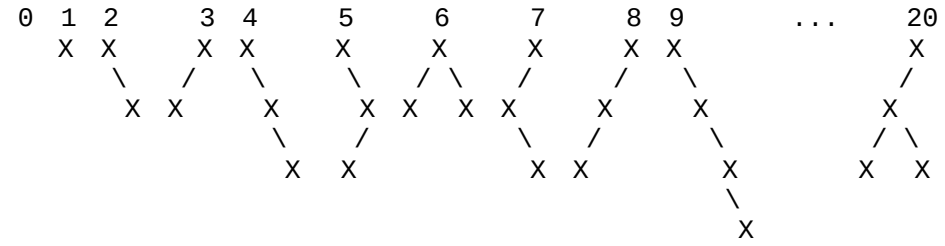
Sample Output
```
Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.
```

Problem D - Trees Made to Order

We can number binary trees using the following scheme:

- The empty tree is numbered 0.
- The single-node tree is numbered 1.
- All binary trees having *m* nodes have numbers less than all those having *m* + 1 nodes.
- Any binary tree having *m* nodes with left and right subtrees *L* and *R* is numbered *n* such that all trees having *m* nodes numbered greater than *n* have either
  - a left subtrees numbered higher than *L*, or
  - a left subtree = *L* and a right subtree numbered higher than *R*.

The first 10 binary trees and tree number 20 in this sequence are shown below:



Your job for this problem is to output a binary tree when given its order number.

Input
Input consists of multiple problem instances. Each instance consists of a single integer *n*, where $1 \leq n \leq 500,000,000$. A value of $n = 0$ terminates input. (Note that this means you will never have to output the empty tree.) There will be at most 200 test cases.

Output
For each problem instance, you should output one line containing the tree corresponding to the order number for that instance. To print out the tree, use the following scheme:

- A tree with no children should be output as `X`.
- A tree with left and right subtrees *L* and *R* should be output as `(`*L'*`)X(`*R*'`)`, where *L'* and *R'* are the representations of *L* and *R*.
  - If L is empty, just output `X(`*R*'`)`.
  - If R is empty, just output `(`*L'*`)X`.

Sample input
```
1
20
31117532
0
```

Sample output
```
X
((X)X(X))X
(X(X(((X(X))X(X))X(X))))X(((X((X)X((X)X)))X)X)
```

Problem E –No Change

Though it might be hard to imagine, the inhabitants of a small country Additivia do not know of such thing as change, which probably has to do with them not knowing subtraction either. When they buy something, they always need to have the exact amount of addollars, their currency. The only other option, but not a really attractive one, is over-paying.

Professor Adem, one of the Additivian mathematicians came up with an algorithm for keeping a balanced portfolio. The idea is the following. Suppose you have more coins of value v_1 than coins of value v_2. In this case you should try to spend at least as many coins of value v_1 as those of value v_2 on any buy you make. Of course spending too many v_1 coins is not a good idea either, but to make the algorithm simpler professor Adem decided to ignore the problem. The algorithm became an instant hit and professor Adem is now designing a kind of “electronic portfolio” with built-in Adem's algorithm. All he needs now is a software for these machines, that will decide whether a given amount of *addollars* can be paid using a given set of coins according to the rules of Adem's algorithm. Needless to say, you are his chosen programmer for the task.

Write a program that reads the description of a set of coins and an amount of *addollars* to be paid, and determines whether you can pay that amount according to Professor Adem's rules.

Input
The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. The input starts with the amount of addollars to be paid x, where $1 \leq x \leq 100000$. The number of different coin values k follows, where $1 \leq k \leq 5$. The values of the coins v_1, ..., v_k follow, where $1 \leq v_i \leq 10000$.

Notice that the order among coin values is significant: you need to spend at least as many coins of value v_1 as coins of value v_2, at least as many coins of value v_2 as those of value v_3, and so on. You may assume that you have a sufficiently large number of coins of each value. There will be at most 100 test cases.

Output
For each test case, your program should output either a single word “`YES`”, if the given amount can be paid according to the rules, or a single word “`NO`” otherwise.

Sample input
```
2
13 3 9 2 1
11 4 1 3 5 7
```

Sample output
```
NO
YES
```

Problem F –Pairs of integers

You are to find all pairs of integers such that their sum is equal to the given integer number *N* and the second number results from the first one by striking out one of its digits. The first integer always has at least two digits and starts with a non-zero digit. The second integer always has one digit less than the first integer and may start with a zero digit.

Input
The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them consists of a single integer *N* where $10 \leq N \leq 1{,}000{,}000{,}000$). There will be at most 200 test cases.

Output
For each test case, the output must follow the description below. On the first line, write the total number of different pairs of integers that satisfy the problem statement. On the following lines, write all those pairs. Write one pair on a line in ascending order of the first integer in the pair. Each pair must be written in the following format:
X + Y = N
Here X, Y, and N, must be replaced with the corresponding integer numbers. There should be exactly one space on both sides of '+' and '=' characters.

Sample input
```
5
10
11
302
1002
14666
```

Sample Onput
```
1
10 + 0 = 10
1
10 + 1 = 11
5
251 + 51 = 302
275 + 27 = 302
276 + 26 = 302
281 + 21 = 302
301 + 01 = 302
4
906 + 96 = 1002
911 + 91 = 1002
951 + 51 = 1002
1001 + 001 = 1002
5
12333 + 2333 = 14666
12833 + 1833 = 14666
13283 + 1383 = 14666
13328 + 1338 = 14666
13333 + 1333 = 14666
```