ANZAC League 2015 Round 3

Problem Set

Problems selected and compiled by Malcolm Corney

Notes:

1) Time limits for each problem are stated.

This page has intentionally been left almost blank

A: Car Troubles

Time Limit: 2 sec

The city center of an unnamed Nordic university town consists of what was once a medieval city with narrow winding streets completely surrounded by a high wall protecting the city against Swedish invaders and other unwanted elements. The wall has since been removed and replaced by a system of interconnecting roads completely circumscribing the old part of the town. The roads inside still remains more or less the same as it was in the middle ages, which of course comes in conflict with modern requirements for accessibility by car, resulting in a maze of twisty little one-way streets, all alike, mixed with slightly wider two-way streets.

Making changes to the traffic routes in such a city can easily cause unexpected side effects if you do not plan carefully ahead. The story goes that a prominent member of the city council once submitted a proposal to the council regarding extensive changes to how the traffic should be organized in the city center. The proposal did have the merit that it would be very easy to drive in to the central square, but it would unfortunately also be impossible to drive out again. The council member in question later went on to become minister of justice in the country under the parole that society should be harder on criminals – "it should be easy to go to jail, but difficult to get out again".

To avoid mistakes as the one above, the city planners need you to develop a tool that can help them discover any traffic problems in the planning stage. The planners need to be alerted of two different situations. The first situation is that a street exists in the city center from which you cannot reach the surrounding, circular, system of roads, i.e., you are *trapped* inside the city. The second situation is that a street exists in the city that cannot be reached from the surrounding system of roads, i.e., it is *unreachable*.

Input

The input consists of a description of how streets connect to each other and the surrounding circular road system. Each street (or a segment of a street) within the city center is represented by an arbitrary integer id number $> 0, (0 < id < 1000)$. The surrounding circular road system is represented by the special id number 0.

First line: An integer giving the number of streets (including the surrounding road system, $0 < streets \leq 1000$).

The following lines: One line for each street (no particular order required and the surrounding road system is included) consisting of a number of integers. First an integer giving the id number of the street. Second, the number of (other) streets that can be reached from this street. Third, a sequence of street id numbers indicating which streets can be reached from this street.

Output

One line for each street on which you would be trapped within the city consisting of the text "`TRAPPED X`" where "`X`" is replaced by street id number in question.

Then, one line for each street within the city that is unreachable from the surrounding system of roads consisting of the text "`UNREACHABLE X`" where `X` should be replaced by the street id in question.

If no problems are found, i.e., you are not trapped in any street and every street is reachable, you should print a single line containing the text "`NO PROBLEMS`".

If multiple streets cause you to get trapped – or are unreachable – you should list them in the same order they were entered in the input (within respective category).

Sample Input and Output

Sample Input 1

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

Sample Output 1

```
TRAPPED 3
UNREACHABLE 4
UNREACHABLE 5
```

Sample Input 2

```
2
1 1 0
0 1 1
```

Sample Output 2

```
NO PROBLEMS
```

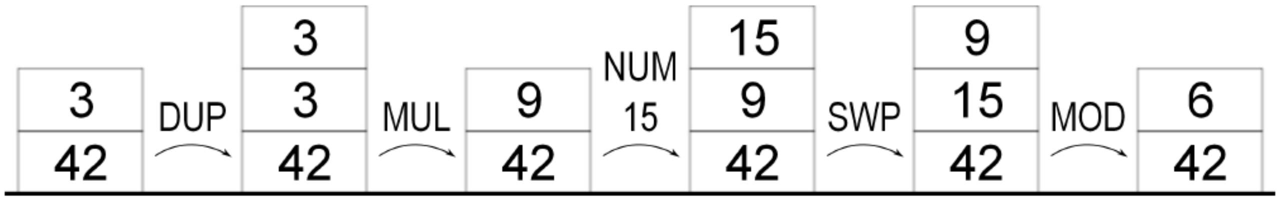
B: Stack Machine Executor

Time Limit: 2 sec

Many calculations can be computed much faster using various machines and automata. In this problem, we will focus on one particular type of machine, called a stack machine. Its name comes from the fact that the machine operates with the well-known data structure stack. The later-stored values are on the top, older values at the bottom. Machine instructions typically manipulate the top of the stack only.

Our stack machine is relatively simple: It works with integer numbers only, it has no storage beside the stack (no registers etc.) and no special input or output devices. The set of instructions is as follows:

- **NUM** X, where X is a non-negative integer number, $0 \leq X \leq 10^9$. The NUM instruction stores the number X on top of the stack. It is the only parametrised instruction.
- **POP**: removes the top number from the stack.
- **INV**: changes the sign of the top-most number ($42 \rightarrow -42$).
- **DUP**: duplicates the top-most number on the stack.
- **SWP**: swaps (exchanges) the position of two top-most numbers.
- **ADD**: adds two numbers on the top of the stack.
- **SUB**: subtracts the top-most number from the "second one" (the one below).
- **MUL**: multiplies two numbers on the top of the stack.
- **DIV**: integer division of two numbers on the top. The top-most number becomes divisor, the one below dividend. The quotient will be stored as the result.
- **MOD**: modulo operation. The operands are the same as for the division but the remainder is stored as the result.



All binary operations consider the top-most number to be the right operand, the second number the "left" one. All of them remove both operands from the stack and place the result on top in place of the original numbers.

If there are not enough numbers on the stack for an instruction (one or two), the execution of such an instruction will result into a program *failure*. A failure also occurs if a divisor becomes zero (for DIV or MOD) or if the result of any operation should be more than 10^9 in absolute value. This means that the machine only operates with numbers between 1 000 000 000 and 1 000 000 000, inclusive.

To avoid ambiguities while working with negative divisors and remainders: If some operand of a division operation is negative, the absolute value of the result should always be computed with absolute values of operands, and the sign is determined as follows: The quotient is negative if (and only if) exactly one of the operands is negative. The remainder has the same sign as the dividend. Thus, 13 div 4 = 3, 13 mod 4 = 1, 13 mod 4 = 1, etc.

If a failure occurs for any reason, the machine stops the execution of the current program and no other instructions are evaluated in that program run.

Input

The input contains descriptions of several machines. Each machine is described by two parts: the program and the input section.

The program is given by a series of instructions, one per line. Every instruction is given by three uppercase letters and there will not be any other characters. The only exception is the NUM instruction, which has exactly one space after the three letters followed by a non-negative integer number between 0 and 10^9. The only allowed instructions are those defined above. Each program is terminated by a line containing the word "`END`" (and nothing else).

The input section starts with an integer $N (0 \leq N \leq 10000)$, the number of program executions. The next N lines contain one number each, specifying an input value $V_i, 0 \leq V_i \leq 10^9$. The program should be executed once for each of these values independently, every execution starting with the stack containing one number - the input value V_i.

There is one empty line at the and of each machine description. The last machine is followed by a line containing the word "`QUIT`". No program will contain more than 100 000 instructions and no program requires more than 1 000 numbers on the stack in any moment during its execution.

Output

For each input value, print one line containing the output value for the corresponding execution, i.e., the one number that will be on the stack after the program executes with the initial stack containing only the input number.

If there is a program failure during the execution or if the stack size is incorrect at the end of the run (either empty or there are more numbers than one), print the word "`ERROR`" instead.

Print one empty line after each machine, including the last one.

Sample Input and Output

Sample Input

```
DUP
MUL
NUM 2
ADD
END
3
1
10
50

NUM 1
NUM 1
ADD
END
2
42
43

NUM 600000000
ADD
END
3
0
600000000
1

QUIT
```

Sample Output

```
3
102
2502

ERROR
ERROR

600000000
ERROR
600000001
```

C: Vignère Cipher Encryption

Time Limit: 2 sec

One of the oldest and most common encryption algorithms is the *Vigenère Cipher*. It is quite an old thing - a similar encryption was first described in 1553 by Giovan Battista Bellaso and improved in 1586 by Blaise de Vigenère.

Vigenère encryption produces a single letter of ciphertext for each letter of plaintext, combining one plaintext letter with one single letter of a key on the corresponding position. If the key is shorter than the plaintext, it is simply repeated as needed, e.g. for a key of length 3 and plaintext of length 7, letters will be combined like this (K_i is the key letter, P_i is the plaintext letter, and C_i is the resulting ciphertext letter).

$$\begin{array}{ccccccc} K_1 & K_2 & K_3 & K_1 & K_2 & K_3 & K_1 \\ P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 \\ \hline C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \end{array}$$

The letter of the key specifies how many positions should be the plaintext letter "shifted forward" in the alphabet. If a key letter is A, the corresponding plaintext letter will be shifted by one character, B means two positions, etc. The alphabet is considered circular, so if the last letter (Z) should be shifted, it becomes A again. Please note that A (key) combined with another A (plaintext) will result in B, which may be a little unusual for the common Vigenère cipher. The Vigenère square at the end of this problem statement gives an overview how letters of a plaintext get combined with letters of a key to produce the ciphertext.

Your task is to write a program that will encrypt messages using the Vigenère cipher with a given key.

Input

The input contains several instances. Each instance consists of two lines, the first line is the encryption key and the second line is the plaintext. Both key and plaintext consist of uppercase letters of the English alphabet $A, B, C, \ldots, Z$. The length of the key will be between 1 and 1000, the length of the plaintext between 1 and 100 000, inclusive.

Input is terminated by a line containing one zero.

Output

For each input instance, output the ciphertext - the encrypted version of the message.

Sample Input and Output

Sample Input	Sample Output
`ICPC`	`CKYVRVIHLUUWVHIVJJU`
`THISISSECRETMESSAGE`	`DHAUUNMHHSRCFSEPJEBPZJQTDRAUHFU`
`ACM`	`OTFJ`
`CENTRALEUROPEPROGRAMMINGCONTEST`	
`LONGKEY`	
`CERC`	
`0`	

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

Vigenère square

Mapping a given plaintext letter (column) and a key letter (row) to the resulting ciphertext letter

D: Farey Sums

Time Limit: 2 sec

Given a positive integer, N, the sequence of all fractions a/b with $(0 < a \leq b)$, $(1 < b \leq N)$ and a and b relatively prime, listed in increasing order, is called the *Farey Sequence* of order N.

For example, the *Farey Sequence of order 6* is:

$$0/1,\ 1/6,\ 1/5,\ 1/4,\ 1/3,\ 2/5,\ 1/2,\ 3/5,\ 2/3,\ 3/4,\ 4/5,\ 5/6,\ 1/1$$

If the denominators of the *Farey Sequence of order* $\boldsymbol{N}$ are:

$$b[1],\ b[2],\ \ldots,\ b[K]$$

then the *Farey Sum* of order N is the sum of $b[i]\ /\ b[i+1]$ from $i = 1$ to $K - 1$.

For example, the *Farey Sum of order 6* is:

$$1/6\ +\ 6/5\ +\ 5/4\ +\ 4/3\ +\ 3/5\ +\ 5/2\ +\ 2/5\ +\ 5/3\ +\ 3/4\ +\ 4/5\ +\ 5/6\ +\ 6/1$$

Write a program to compute the *Farey Sum of order N*.

Input

The first line of input contains a single integer P, $(1 \leq P \leq 10000)$, which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, K, followed by the order N, $(2 \leq N \leq 10000)$, of the *Farey Sum* that is to be computed.

Output

For each data set there is a single line of output. The single output line consists of the data set number, K, followed by a single space followed by the *Farey Sum* as a decimal fraction in lowest terms. If the denominator is 1, print only the numerator.

Sample Input and Output

Sample Input	Sample Output

4
1 6
2 15
3 57
4 9999
	1 35/2
	2 215/2
	3 2999/2
	4 91180457/2

This page has intentionally been left almost blank

E: Refrigerator Magnets

Time Limit: 2 sec

Like many families with small children, my family's refrigerator is adorned with a set of alphabet magnets: 26 separate magnets, each containing one letter of the alphabet. These magnets can be rearranged to create words and phrases. I feel it is my parental duty to use these magnets to create messages that are witty and insightful, yet at the same time caring and supportive. Unfortunately, I am somewhat hindered in this task by the fact that I can only make phrases that use each letter once.

For example, a nice inspiring message to leave for the children might be, "I LOVE YOU." Unfortunately, I cannot make this message using my magnets because it requires two letter "O"s. I can, however, make the message, "I LOVE MUSTARD." Admittedly this message isn't as meaningful, but it does manage to not use any letters more than once.

You are to write a program that will look at a list of possible phrases and report which phrases can be written using refrigerator magnets.

Input

The input will consist of one or more lines, ending with a line that contains only the word "END".

Each line will be 60 characters or less, and will consist of one or more words separated by a single space each, with words using only uppercase letters (A - Z). There will not be any leading or trailing whitespace, and there will not be any blank lines.

Output

Output only the lines which can be written in refrigerator magnets - that is, the lines which have no duplicate letters. Output them exactly the same as they were in the input - white spaces and all. Do not output the final "END" string.

Sample Input and Output

Sample Input	Sample Output

`I LOVE YOU`
`I LOVE MUSTARD`
`HAPPY BIRTHDAY`
`GLAD U BORN`
`SMILE`
`IMAGINE`
`WHATS UP DOC`
`HAVE A NICE DAY`
`END`
	`I LOVE MUSTARD`
	`GLAD U BORN`
	`SMILE`
	`WHATS UP DOC`

This page has intentionally been left almost blank

F: Indiana Jones and the Lost Soccer Cup

Time Limit: 5 sec

In 1930 the first FIFA World Cup was held in Uruguay, and Uruguay won the Cup in a dramatic final against Argentina. All of this seems like ages past, and now myth and legend rank around that historic first World Cup. Some even claim that the original trophy cup that was awarded to Uruguay for their victory has mythical powers and would grant any side the strength to win the World Cup. Now that original cup has been lost in time, and even though its mythical powers are probably just stories, it is still an important artefact which belongs in a museum. Clearly an expert is needed to recover it.

Famous archaeologist and adventurer Indiana Jones has taken this dangerous task on himself and travelled to Uruguay to find the cup. His search has led him to an ancient underground cave system where the cup is rumoured to have been hidden. Many traps lure within these caves, and only his instinct and his faithful whip have saved Indy from certain death. Now he has reached a mysterious enormous gate and he can only speculate that the cup must be hidden behind that gate. Unfortunately it is closed shut.

The gate is riddled with switches and levers, and all of them are denoted with letters and numbers. As you may have guessed, the gate will only open if the switches and levers are pulled in the correct order, but beware! – for if anyone is unlucky enough to get the order wrong, doom awaits him. Luckily, during his exploration of the caves Indy has found several encrypted hints which provide clues about the correct sequence. Here's one: "The faithful knows that X comes before O"; and another: "Under no circumstances should you touch Δ unless the Θ has been moved!" Clearly these clues give hints about the correct order, but there are a lot of switches and levers, and there are lots of clues. Indy needs help!

Given all of the hints Indy has collected, can you help him determine the correct order of the levers and switches so that he can successfully complete his adventure? But beware – Indy could have missed some hints; or perhaps he misinterpreted some of them. The former case will likely leave more than one possible sequence while the latter will lead to no possible sequence at all. You must detect these cases and warn Indy.

Input

The first line of input contains the number of test cases C, $(C \leq 30)$. For each test case there is one line with the number n, $(1 \leq n \leq 10\,000)$ of switches/levers on the gate and the number h, $(0 \leq h \leq 100\,000)$ of hints that Indy has discovered. Then follow h lines, one for each clue, with the numbers a and b $(1 \leq a,\ b \leq n,\ a \neq b)$, meaning that lever a must be pulled before lever b.

Output

For each test case output one line with the correct sequence of the numbers 1 to n. Separate the numbers with a single space. If there is no possible sequence, print "`recheck hints`" instead. If there are multiple possible sequences, print "`missing hints`" instead.

Sample Input and Output

Sample Input	Sample Output

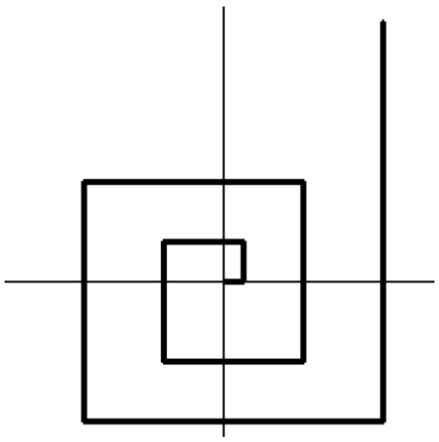
3
3 2
1 2
3 1
3 1
1 2
3 2
1 2
2 1
	3 1 2
	missing hints
	recheck hints

This page has intentionally been left almost blank

G: Growing Rectangular Spiral

Time Limit: 2 sec

A growing rectangular spiral is a connected sequence of straight-line segments starting at the origin. The first segment goes right (positive `x` direction). The next segment goes up (positive `y` direction). The next segment goes left (negative `x` direction). The next segment goes down (negative `y` direction) and the sequence of directions repeats. Each segment has integer length and each segment is at least one unit longer than the previous segment. In the spiral below, the segment lengths are 1, 2, 4, 6, 7, 9, 11, 12, 15, 20.



Write a program to determine the shortest growing rectangular spiral (in total length) that ends at a given integer point `(x,y)` in the first quadrant or determine that there is no such spiral.

Input

The first line of input contains a single integer P, $(1 \leq P \leq 1\,000)$, which is the number of data sets that follow. Each dta set should be processed identically and independently.

Each data set consists of a single line of input consisting of three space separated decimal integers. The first integer is the data set number. The next two integers are the x and y coordinates of the desired end point $(1 \leq x \leq 10\,000, 1 \leq y \leq 10\,000)$.

Output

For each data set there is a single line of output. If there is no spiral solution, the line consists of the data set number, a single space and "`NO PATH`" (without the quotes. If there is a solution, the line consists of the data set number, a single space, the number of segments in the solution, a single space, followed by the lengths of the segments in order, separated by single spaces. The input data will be chosen so that no path requires more than 22 segments.

Sample Input and Output

Sample Input	Sample Output

`3`
`1 1 1`
`2 3 5`
`3 8 4`
	`1 NO PATH`
	`2 2 3 5`
	`3 6 1 2 3 9 10 11`

This page has intentionally been left almost blank

H: Happy Happy Prime Prime

Time Limit: 2 sec

RILEY VASHTEE: [*reading from display*] Find the next number in the sequence:

`313 331 367 ...?  What?`

THE DOCTOR: 379.

MARTHA JONES: What?

THE DOCTOR: It's a sequence of happy primes – 379.

MARTHA JONES: Happy *what*?

THE DOCTOR: Any number that reduces to one when you take the sum of the square of its digits and continue iterating it until it yields 1 is a happy number. Any number that doesn't, isn't. A *happy prime* is both happy and prime.

THE DOCTOR: I dunno, talk about *dumbing* down. Don't they teach recreational mathematics anymore?

Excerpted from *"Dr. Who"*, Episode 42 (2007).

The number 7 is certainly prime. But is it happy?

$$\begin{array}{rcl} 7 & \rightarrow & 7^2 = 49 \\ 49 & \rightarrow & 4^2 + 9^2 = 97 \\ 97 & \rightarrow & 9^2 + 7^2 = 130 \\ 130 & \rightarrow & 1^2 + 3^2 = 10 \\ 10 & \rightarrow & 1^2 + 0^2 = 1 \end{array}$$

It is happy ☺. As it happens, 7 is the smallest happy prime. Please note that for the purposes of this problem, 1 is *not* prime.

For this problem you will write a program to determine if a number is a *happy prime.*

Input

The first line of input contains a single integer P, $(1 \le P \le 1\,000)$, which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, K, followed by the happy prime candidate, m, $(1 \le m \le 10\,000)$.

Output

For each data set there is a single line of output. The single output line consists of the data set number, K, followed by a single space followed by the candidate, m, followed by a single space, followed by **YES** or **NO**, indicating whether m is a happy prime.

Sample Input and Output

Sample Input	Sample Output

4
1 1
2 7
3 383
4 1000
	1 1 NO
	2 7 YES
	3 383 YES
	4 1000 NO

This page has intentionally been left almost blank

I: Lucky Digit

Time Limit: 2 sec

John believes that the digit D is lucky, and looks out for it in numbers everywhere. When he learned how to represent numbers of bases other than 10, he was very excited, because some numbers will be luckier in other bases. He wants your help to determine, for each number he gives you, the luckiest representation in any base from 2 to 10, inclusive.

For example, suppose $D = 7$, and John gives you the number 507 (in base 10). In base 8 this would be written 773 (because $7 \cdot 8^2 + 7 \cdot 8^1 + 3 \cdot 8^0 = 507$), and in base 9 it would be written 623. Since 773 has the most 7s in it, this is considered the luckiest representation. If there is a tie, the representation in the higher base is considered luckier.

Input

The input consists of an arbitrary number of records, but no more than 50. Each record is a line containing two integers (in base 10), N and D, separated by a space. The end of input is marked by a line containing only the value -1.

In all test cases, $0 \leq N \leq 1\,000\,000$ and $0 \leq D \leq 9$.

Output

For each test case, output a line containing the luckiest representation of N for the lucky digit D.

Sample Input and Output

Sample Input	Sample Output

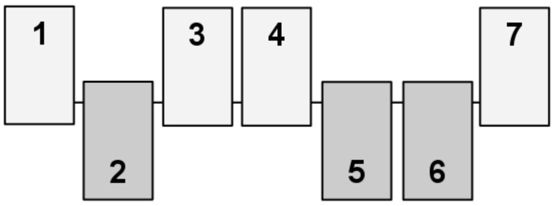
`507 7`
`64 0`
`123 9`
`-1`
	`773`
	`1000000`
	`123`

This page has intentionally been left almost blank

J: Shut the Box

Time Limit: 2 sec

Shut the Box is a one-player game that begins with a set of N pieces labelled from 1 to N. All pieces are initially "unmarked" (in the picture below, the unmarked pieces are those in an upward position). In the version we consider, a player is allowed up to T turns, with each turn defined by an independently chosen value V (typically determined by rolling one or more dice). During a turn, the player must designate a set of currently unmarked pieces whose numeric labels add precisely to V, and mark them. The game continues either until the player runs out of turns, or until a single turn when it becomes impossible to find a set of unmarked pieces summing to the designated value V (in which case it and all further turns are forfeited). The goal is to mark as many pieces as possible; marking all pieces is known as "shutting the box". Your goal is to determine the maximum number of pieces that can be marked by a fixed sequence of turns.



As an example, consider a game with 6 pieces and the following sequence of turns: 10, 3, 4, 2. The best outcome for that sequence is to mark a total of four pieces. This can be achieved by using the value 10 to mark the pieces 1+4+5, and then using the value of 3 to mark piece 3. At that point, the game would end as there is no way to precisely use the turn with value 4 (the final turn of value 2 must be forfeited as well). An alternate strategy for achieving the same number of marked pieces would be to use the value 10 to mark four pieces 1+2+3+4, with the game ending on the turn with value 3. But there does not exist any way to mark five or more pieces with that sequence.

Input

Each game begins with a line containing two integers, N, T where $1 \leq N \leq 22$ represents the number of pieces, and $1 \leq T \leq N$ represents the maximum number of turns that will be allowed. The following line contains T integers designating the sequence of turn values for the game; each such value V will satisfy $1 \leq V \leq 22$. You must read that entire sequence from the input, even though a particular game might end on an unsuccessful turn prior to the end of the sequence. The data set ends with a line containing 0 0.

Output

You should output a single line for each game, as shown below, reporting the ordinal for the game and the maximum number of pieces that can be marked during that game.

Sample Input and Output

Sample Input

```
6 4
10 3 4 2
6 5
10 2 4 5 3
10 10
1 1 3 4 5 6 7 8 9 10
22 22
22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
0 0
```

Sample Output

```
Game 1: 4
Game 2: 6
Game 3: 1
Game 4: 22
```

This page has intentionally been left almost blank

K: Periodic, My Dear Watson

Time Limit: 2 sec

Given the periodic table and a series of words, determine whether the word can be formed using the elements in the periodic table. If there are multiple ways of doing so, select the one with the fewest number of required elements. If multiple options use the same minimal number of elements, sum the atomic numbers (atomic numbers are implied by the order in which the elements are read in) used for each option, and select the one with the smallest value. If there are still multiple, then there's probably no point, as it is most likely too obvious, so in these cases, simply print "Too Obvious" for the given word.

An 'element' always starts with a capital letter followed by zero or more lower case letters.

Input

Input will begin with a line containing a single integer, N representing the number of cases to process. Each case will being with a line with a single integer, O, representing the number of symbols in the case. The next O lines will contain one element per line. The next line will contain a single integer, P representing the number of words to process in this case. The next P lines contain one word each. O and P as well as the length of any element or name will be less than 5 000.

```
<Number of cases>
<number of elements>
<elements...>
<number of words>
<words...>
```

Output

If a given word has a deterministic answer based on the periodic table provided and the above stated rules, print the case number and the series of elements used, forming the word, but with square brackets around them to indicate the elements. One example output line might be: "[Ba][Na][Na]".

Sample Input and Output

Sample Input	Sample Output
1 4 Ba Na N A 2 banana nan	[Ba][Na][Na] [Na][N]

This page has intentionally been left almost blank