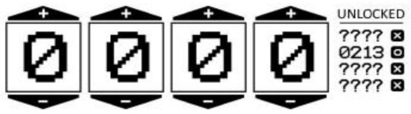
A - Anti-brute Force Lock

Source file name: `anti.py`
Time limit: 1 second

Lately, there is one serious problem with Panda Land Safe Box: several safes have been robbed! The safes are using old 4-digits rolling lock combination (you only have to roll the digit, either up or down, until all four of them match the key). Each digit is designed to roll from 0 to 9. Rolling-up at 9 will make the digit become 0, and rolling-down at 0 will make the digit become 9. Since there are only 10000 possible keys, 0000 through 9999, anyone can try all the combinations until the safe is unlocked.



What's done is done. But in order to slow down future robbers' attack, Panda Security Agency (PSA) has devised a new safer lock with multiple keys. Instead of using only one key combination as the key, the lock now can have up to *N* keys which has to be all unlocked before the safe can be opened. These locks works as the following:

- Initially the digits are at 0000.
- Keys can be unlocked in any order, by setting the digits in the lock to match the desired key and then pressing the UNLOCK button.
- A magic JUMP button can turn the digits into any of the unlocked keys without doing any rolling.
- The safe will be unlocked if and only if all the keys are unlocked in a minimum total amount of rolling, excluding JUMP (yes, this feature is the coolest one).
- If the number of rolling is exceeded, then the digits will be reset to 0000 and all the keys will be locked again. In other word, the state of the lock will be reset the cracking is failed.

For example, if the keys to be unlocked are 1111, 1155, and 5511, then this can be done in 20 turns:

- Turn 0000 into 1111, rolls: 4.
- Turn 1111 into 1155, rolls: 8.
- Jump 1155 into 1111, we can do this because 1111 has been unlocked before.
- Turn 1111 into 5511, rolls: 8.

PSA is quite confident that this new system will slow down the cracking, giving them enough time to identify and catch the robbers. In order to determine the minimum number of rolling needed, PSA wants you to write a program. Given all the keys, calculate the minimum number of rolls needed to unlock the safe.

Input

The first line of input contains an integer T, the number of test cases follow. Each case begins with an integer N ($1 \leq N \leq 500$), the number of keys. Then follow N four digits numbers (leading zero allowed) representing the keys to be unlocked. Numbers are separated by blanks.

The input must be read from standard input.

Output

For each case, print in a single line the minimum number of rolls needed to unlock all the keys.

The output must be written to standard output.

Sample Input

```
4
2 1155 2211
3 1111 1155 5511
3 1234 5678 9090
4 2145 0213 9113 8113
```

Sample Output

```
16
20
26
17
```

B - Helping Fill Bates

Source file name: `bates.py`
Time limit: 1 second

Everyone knows Fill Bates but I guess fewer people know that he was famous in his high school days for a different reason. I just put below the exact lines from one of his short biographies:

> Before high-school graduation, Bates was renowned for designing the class scheduling soft-ware that placed him in a class full of girls. With his present success, those girls are probablymutilating themselves for not buying in on an early investment.

Now you are to help him with a completely different purpose. His company has initiated a talent search program in QSA. It has 52 states (the original 50 states plus the two recent troublesome additions Oraq and Malistan). The candidates from the 52 states are given a state identity and a serial number (The serial number is unique). The state identities for the 52 states are the 52 ASCII characters A..Z and a..z. The talent search process is a bit strange. At most 1 million candidates stand in a single line according to the increasing order their serial number (Starting with 0 and then 1, 2, 3 etc) with a placard showing only their state identity (After all who wants to hire employees from Oraq and Malistan). Mr. Fill Gates then writes a sequence of characters (Only alphabets). If candidates are found in that order of states with increasing serial numbers (Some candidates may be skipped for this purpose) then those candidates are taken. Other wise an appropriate message is given.

Input

The input file contains only one set of input. First line of the input file contains a string S containing only alphabets. The length of this string is at most 1 000 000. The next line contains an integer Q ($0 < Q < 3\,501$) which indicates the number of queries. Each of the next Q lines contain one string SS of length less than 10001. These strings are the strings written by Fill Bates.

The input must be read from standard input.

Output

For each query you should output one line. If candidates are not found in the order written by Fill Bates then you should output a string '`Not matched`' (Without the quotes), otherwise you should print '`Matched`' (Note that an space is printed after '`Matched`') and then the serial of the first candidate in the subsequence and the serial of the last candidate of the subsequence. These two integers should be separated by a single space. If there is more than one such subsequence then choose the one which has smallest starting serial number. If there is a tie choose the one with the smallest ending serial number.

The output must be written to standard output.

Sample Input	Sample Output

`aaaaaaaaaaaaaabbbbbbbbbdddddddddddccccccccccccc`
`3`
`aaaaaaaaaaaaaaaaaaa`
`aaaaaaaaaaabbbbbbbbbbbc`
`abccc`
	`Not matched`
	`Not matched`
	`Matched 0 36`

C - Cube Painting

Source file name: `cube.py`
Time limit: 1 second

We have a machine for painting cubes. It is supplied with three different colors: blue, red and green. Each face of the cube gets one of these colors. The cube's faces are numbered as in Figure 1.

Since a cube has 6 faces, our machine can paint a face-numbered cube in $3^6 = 729$ different ways. When ignoring the face-numbers, the number of different paintings is much less, because a cube can be rotated. See example below.

We denote a painted cube by a string of 6 characters, where each character is a '`b`', '`r`', or '`g`'. The i-th character ($1 \le i \le 6$) from the left gives the color of face i. For example, Figure 2 is a picture of "`rbgggr`" and Figure 3 corresponds to "`rggbgr`". Notice that both cubes are painted in the same way: by rotating it around the vertical axis by 90°, the one changes into the other.

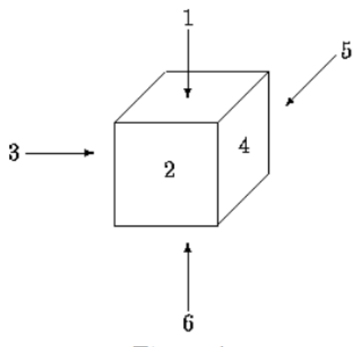


Figure 1

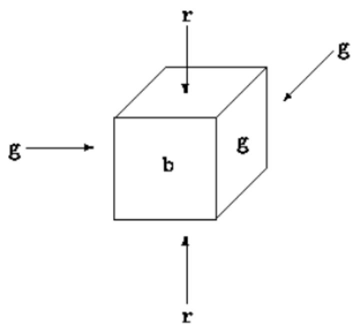


Figure 2

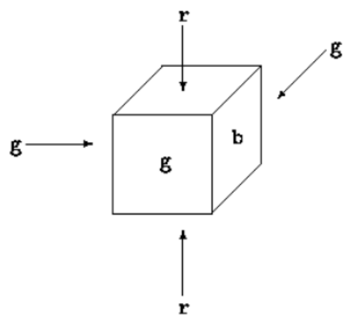


Figure 3

Input

The input of your program is a text file that ends with the standard end-of-file marker. Each line is a string of 12 characters. The first 6 characters of this string are the representation of a painted cube, the remaining 6 characters give you the representation of another cube. Your program determines whether these two cubes are painted in the same way, that is, whether by any combination of rotations one can be turned into the other. (Reflections are not allowed).

The input must be read from standard input.

Output

The output is a file of boolean. For each line of input, output contains '`TRUE`' if the second half can be obtained from the first half by rotation as describes above, '`FALSE`' otherwise.

The output must be written to standard output.

Sample Input	Sample Output

`rbgggrrggbgr`
`rrrbbbrrbbbr`
`rbgrbgrrrrrg`
	`TRUE`
	`FALSE`
	`FALSE`

D - The Dominoes Solitaire

Source file name: `dominoes.py`
Time limit: 1 second

A man used to play dominoes with some friends in a small town. Some families have left the town and at present the only residents in the town are the man and his wife. This man would like to continue playing dominoes, but his wife does not like plaing. He has invented a game to play alone. Two pieces are picked out and are put at the two extremes of a row with a number (n) of spaces. After that, other pieces (m) are picked out to fill the spaces. The number of pieces is greater than or equal to the number of spaces ($m \geq n$), and the number of pieces is less than or equal to 14 ($m \leq 14$). The spaces are filled by putting one piece in each space, accoding to the rules of dominoes: the number of adjacent dots on two different dominoes must coincide. Pieces with repeated values are placed in the same way as the other pieces, and not at right angels.

The problem consists in the design of a program which, given a number of spaces (n), a number of pieces (m), the two initial pieces (i_1, i_2) and (d_1, d_2), and the m pieces $(p_1, q_1), (p_2, q_2), \ldots, (p_m, q_m)$, decides if it is possible to fill the n spaces between the two initial pieces using the m pieces and with the rules of dominoes. For example, with $n = 3$, $m = 4$, initial pieces $(0, 1)$ and $(3, 4)$, and pieces $(2, 1), (5, 6), (2, 2)$ and $(3, 2)$, the answer is '`YES`', because there is a solution: $(0, 1), (1, 2), (2, 2), (2, 3), (3, 4)$. With $n = 2$, $m = 4$, pieces in the extremes $(0, 1)$ and $(3, 4)$, and $(1, 4), (4, 4), (3, 2)$ and $(5, 6)$, the answer is '`NO`'.

Input

The input will consist of a series of problems, with each problem described in a series of lines: in the first line the number of spaces (n) is indicated, in the second line the number of pieces (m) used to fill the spaces, in the next line the piece to be placed on the left, with the two values in the piece separated by a space and in the same way that the numbers appear in the row, in the following lines the piece to be placed on the right, with the two values in the piece separated by a space and in the same way that the numbers appear in the row, and the other m pieces appear in consecutive lines, one in each line, with the two values separated by a space. Different problems appear in the input successively without separation, and the input finishes when '`0`' appears as the number of spaces.

The input must be read from standard input.

Output

For each problem in the input a line is written, with '`YES`' if the problem has a solution, and '`NO`' if it has no solution.

The output must be written to standard output.

Sample Input	Sample Output

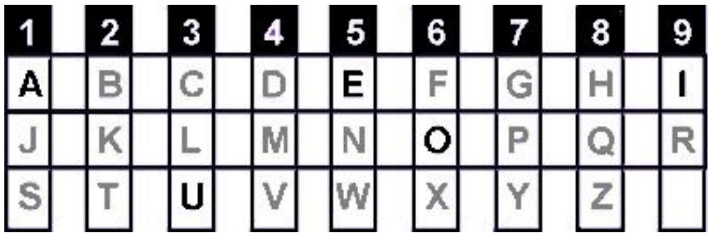
```
3
4
0 1
3 4
2 1
5 6
2 2
3 2
2
4
0 1
3 4
1 4
4 4
3 2
5 6
0
```
	```
	YES
	NO
	```

E - The Mad Numerologist

Source file name: `mad.py`
Time limit: 1 second

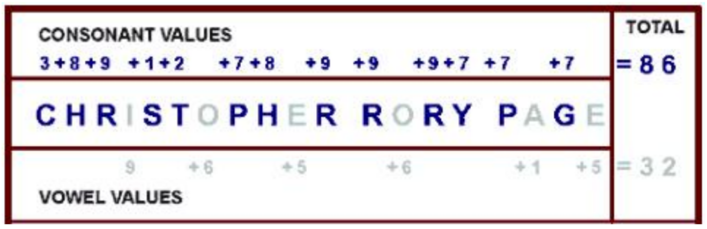
Numerology is a science that is used by many people to find out a mans personality, sole purpose of life, desires to experience etc. Some calculations of numerology are very complex, while others are quite simple. You can sit alone at home and do these easy calculations without taking anyone's help. However, in this problem you won't be asked to find the value of your name.

To find the value of a name, modern numerologists have assigned values to all the letters of English alphabet. The first table below shows the numerical values of all letters of the English alphabet. The five letters A, E, I, O, U are vowels, and the rests of the letters are consonant. In this table, all letters in column 1 have value 1, all letters in column 2 have value 2 and so on. So T has value 2, F has value 6, R has value 9, O has value 6 etc. When calculating the value of a particular name, the consonants and vowels are calculated separately.



1	2	3	4	5	6	7	8	9
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	

The following picture explains this method using the name "CHRISTOPHER RORY PAGE".



So you can see that to find the consonant value, the values of individual consonants are added and to find the vowel value the values of individual vowels are added.

A mad Numerologist suggests people many strange lucky names. He follows the rules stated below while giving lucky names.

- The name has a predefined length N.
- The vowel value and consonant value of the name must be kept minimum.
- To make the pronunciation of the name possible vowels and consonants are placed in alternate positions. Actually vowels are put in odd positions and consonants are put in even positions. The leftmost letter of a name has position 1; the position right to it is position 2 and so on.
- No consonants can be used in a name more than five times and no vowels can be used in a name more than twenty-one times.
- Following the rules and limitations above the name must be kept lexicographically smallest. Please note that the numerologists first priority is to keep the vowel and consonant value minimum and then to make the name lexicographically smallest.

Input

First line of the input file contains an integer N ($0 < N$) that indicates how many sets of inputs are there. Each of the next N lines contains a single set of input. The description of each set is given in a line containing an integer n ($0 < n < 211$) that indicates the predefined length of the name.

The input must be read from standard input.

Output

For each set of input produce one line of output. This line contains the serial of output followed by the name that the numerologist would suggest following the rules above. All letters in the output should be uppercase English letters.

The output must be written to standard output.

Sample Input	Sample Output

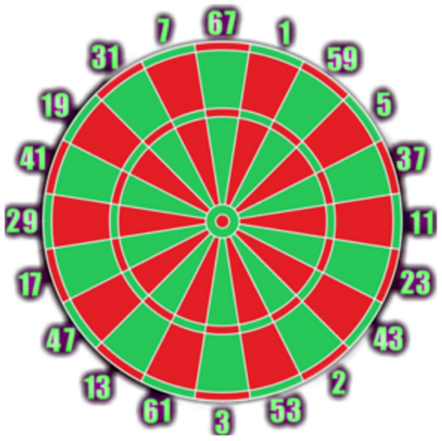
```
3
1
5
5
```
	```
	Case 1: A
	Case 2: AJAJA
	Case 3: AJAJA
	```

F - Prime Darts

Source file name: `prime.py`
Time limit: 1 second

A dartboard manufacturer wants to revolutionize the game of darts, creating a *prime* dart board for math geeks. He has designed several boards with different numbers of areas, so that a board with n areas has the following scores: the first area is worth 1 point, the remaining $n-1$ areas have a value corresponding to the first $n-1$ prime numbers.

For example, a prime dartboard with 20 areas could be as in the Figure.



We want to know the minimum number of darts needed to obtain a score of q points on a prime dartboard of size n.

Input

The first line of the input contains an integer, t, indicating the number of prime dartboards.

For each case, there is a line with two numbers separated by a space. The first one, n, represents the number of areas of the board, with $1 \leq n \leq 100$, and the second number, q, indicates the score we have to get, with $1 \leq q \leq 5000$.

The input must be read from standard input.

Output

For each test case, the output should consist of one line showing the minimum number of darts needed to obtain a q points on a prime dartboard of size n.

The output must be written to standard output.

Sample Input	**Sample Output**

`6`
`1 200`
`5 15`
`5 34`
`6 34`
`7 4`
`20 1000`
	`200`
	`3`
	`6`
	`4`
	`2`
	`16`

G - Travel

Source file name: `travel.py`
Time limit: 1 second

Xiao Ming, a clever and lazy boy, is good at programming and sleeping. He hated traveling by train very much. Nevertheless, tomorrow in the early morning hours he will travel from Liuzhou to Xi'an to in order to get to the National Olympiad of Informatics in 2001 (NOI2001). Since he is afraid of arriving too late and being excluded from the contest, he is looking for the train which gets her to Xi'an as early as possible. However he dislikes getting to the station too early, so if there are several schedules with the same arrival time, he will choose the one with the latest departure time.

Xiao Ming asks you to help him with his problem, so that she can sleep a bit longer tomorrow. You are given a set of railroad schedules from which you have to compute the fastest connection among those with the earliest arrival time for going from one location to another. One good thing: Xiao Ming can switch trains in zero time.

Input

The input file contains several scenarios. Each of them consists of 3 parts:

- Part one lists the names of all cities connected by the railroads. It starts with a line containing an integer C ($1 \leq C \leq 100$) followed by C lines containing city names. These names consist of at most 20 letters.
- Part two describes all the trains running during the day. It start with a number $0 \leq T \leq 100$ followed by T train descriptions. Each train description consists of one line with a number $2 \leq t_i < 100$ and t_i more lines with a time and a city name, meaning that passengers can get on or off the train at that time at that city. The times are given in the 24-hour format 'hhmm'.
- Part three consists of three lines: line one contains the earliest possible starting time for the journey, line two the name of the city where he starts, and line three the destination city. The two cities are always different.

The end of the input file is marked by a line containing only a zero (instead of C). Do not process this line.

The input must be read from standard input.

Output

For each scenario print one line solution. If a connection exists the print 2 four-digit numbers, meaning 'hhmm', the starting time and the arriving time. If not connection exists print a line containing '`No connection`'.

The output must be written to standard output.

Sample Input

```
3
Liuzhou
Guilin
Xian
3
2
0900 Liuzhou
1200 Guilin
2
1200 Guilin
2200 Xian
3
0900 Liuzhou
1200 Guilin
2300 Xian
0800
Liuzhou
Xian
3
Liuzhou
Guilin
Xian
1
3
0900 Liuzhou
1200 Guilin
2300 Xian
1000
Liuzhou
Xian
0
```

Sample Output

```
0900 2200
No connection
```