Labor Strike

There are `n` labor unions in New York City. You have learned that the labor union `i` will undertake a strike every `s_i` days.

For example, assume the 1st day is Sunday, and labor union 1 undertakes a strike ever 4 days. Then there will be a strike on the 4th day (the Wednesday), the 8th day (following Sunday) and so forth. (See the table below.)

Because the MTA employees support all the unions, the subways do not work if at least one of the unions is on strike. The only exception to this are Fridays and Saturdays: even if a union has a planned strike for that day, the subways will still operate as usual.

You need to figure out the number of days that you can not take the subway in the following `m` days.

You can assume the 1st day is always Sunday.

Days	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
Union 1				x				x				x		
Union 2			x			x			x			x		
Union 3								x						
No subway			x	x				x	x			x		

Input

The 1st line contains an integer `m` (`7 <= m <= 3650`) equal to the number of days for which you wish the make the calculation.

The next line contains another integer `n` (`1 <= n <= 100`) representing the number of labor unions.

The *i*-th of the next `n` lines contain a positive integer `s_i` (which will never be a multiple of 7) giving the striking period of labor union *i*.

Output

Output one integer followed by a newline: the total number of days with strikes.

Example 1 (shown in the table above)

```
Input:
14
3
4
3
8

Output:
5
```

You can find there will be strikes on the 3rd, 4th, 6th, 8th, 9th, and 12th days. So the subway will not operate on the 3rd, 4th, 8th, 9th, and 12th days (since the 6th day is a Friday).

Example 2

```
Input:
100
4
12
15
25
40

Output:
15
```

AWESOME Number

We say a number is AWESOME if it is prime and has 1 in its ones place. You are given a number N and you are asked to answer how many AWESOME numbers are not greater than N.

Input The input consists of a single integer N (1 <= N <= 20,000,000).

Output You should print **one line** containing a single integer, the number of awesome numbers that are no larger than N.

Example 1

```
Input:
6

Output:
0
```

Example 2

```
Input:
11

Output:
1
```

Because 11 is AWESOME.

Example 3

```
Input:
100

Output:
5
```

Because 11, 31, 41, 61, and 71 are all AWESOME.

Task Planning

There are two types of tasks: repeating tasks and one-time tasks. Both types of tasks have a start time and an end time. Additionally, repeating tasks have a repetition interval. Repeating tasks keep repeating forever at a fixed frequency. For example, a repeating task with start time 5, end time 8 and repetition interval 100 will occur at time [5…8], [105…108], [205…208], …

You are given N one-time tasks and M repeating tasks and you need to determine if there is any overlap between them in the time interval [0…1000000].

Note: tasks are considered to overlap only if their time intervals overlap, but not if the endpoints are the same. For example, [2…5] and [4…6] are considered to overlap while [2…4] and [4…6] are not.

Input

The first line of the input contains two integers N and M (0 <= N, M <= 100), indicating the number of one-time tasks and repeating tasks, respectively. Each of the following N lines contains two integers indicating the start time and the end time for the one-time tasks. Afterward, each of the following M lines contains three integers indicating the start time, end time and repetition interval of the repeating tasks.

It is guaranteed that all integers are between 0 and 1,000,000, the end time is larger than the start time and the repetition interval is positive.

Output

Print one line either containing `NO CONFLICT` if there is no overlap, or `CONFLICT` if there is at least one overlap.

Example 1

```
Input:
2 0
10 20
20 30

Output:
NO CONFLICT
```

Example 2

```
Input:
2 0
10 30
20 21

Output:
CONFLICT
```

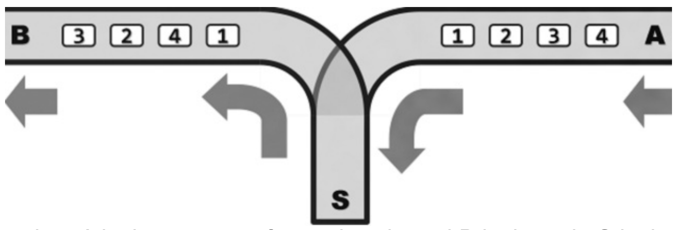
Example 3

```
Input:
1 1
1000 2000
0 10 1000

Output:
CONFLICT
```

Train

This is a figure that shows the structure of a station for train dispatching.



In this station, A is the entrance for each train and B is the exit. S is the switching track. The coaches of a train can enter the switching track from direction A and must leave in direction B. Individual coaches can be disconnected from the rest of the train as they enter the switching track, so that they can be reorganized before they continue in direction B. If a coach enters the switching track from direction A, it must leave in direction B (i.e., it cannot return towards A). If a coach leaves in direction B, it cannot return back to the switching track.

Assume that a train consist of n coaches labeled {1, 2, …, n}. A dispatcher wants to know whether these coaches can pull out at B in the order of {a1, a2, …, an}.

Input

The 1st line contains an integer n (n <= 1,000) equal to the number of coaches, as described above. In each of the next lines of the input, except the last one, there is a permutation of 1, 2, . . . , n, this is the sequence {a1, a2, …, an} that the dispatcher would like to achieve as the coaches leave the switching track in direction B. The last line of the block contains just '0' (to indicate the end of input).

Output

You should output the result for each permutation. If the sequence is feasible, output a "Yes", followed by a newline. If the sequence is infeasible, output a "No", followed by a newline.

Example 1 (pictured in the figure)

```
Input:
4
3 2 4 1
0

Output:
Yes
```

Example 2

```
Input:
5
1 2 3 4 5
4 5 1 3 2
0

Output:
Yes
No
```

Stack Puzzle

There are two sequences of stack operations converting the word TROT to TORT:

```
[
i i i i o o o o
i o i i o o i o
]
```

where `i` and `o` stands for *in* / push and *out* / pop operation respectively. In this problem, you are given two words and you are asked to find out all sequences of stack operations converting the first word to the second word.

Input The input consists of two lines, the first of which is the source word and the second is the target word.

Output Your program should print a sorted list of valid `i/o` sequences. The list is delimited by

```
[
]
```

and the sequences are sorted in lexicographical order. Within each sequence, `i`'s' and `o`'s are separated by a single space and each sequence is terminated by a new line. There should be a newline character printed after the closing square bracket delimiter.

Process Given an input word, a valid `i/o` sequence implies that every character of the word is pushed and popped exactly once, and no attempt is ever made to pop an empty stack. For example, if the word FOO is input, then the sequence:

- `i i o i o o` is valid and produces OFF
- `i i o` is not valid (too short),
- `i i o o o i` is not valid (illegal pop from an empty stack)

A valid sequence results in a permutation of the letters in the input word. For example, given the input word FOO, both sequences `i i o i o o` and `i i i o o o` give the word OOF.

Example 1

```
Input:
madam
adamm

Output:
[
i i i i o o o i o o
i i i i o o o o i o
i i o i o i o i o o
i i o i o i o o i o
]
```

Example 2

```
Input:
bahama
bahama
```

```
Output:
[
i o i i i o o i i o o o
i o i i i o o o i o i o
i o i o i o i i i o o o
i o i o i o i o i o i o
]
```

Example 3

```
Input:
long
short

Output:
[
]
```

Example 4

```
Input:
eric
rice

Output:
[
i i o i o i o o
]
```