Prime Pair

A prime pair is a pair of the form (p, p + 2) where p and (p + 2) are both prime. The first few prime pairs are (3, 5), (5, 7), (11, 13). In this problem, you are given an integer N and you should find the N'th prime pair.

Input The input will contain a single integer N (1 <= N <= 100,000).

Output The output should be a single **line**, the N'th prime pair formatted as `(p1, p2)` (there is a space after the comma). It is guaranteed that primes in the 100,000th prime pair are less than 20,000,000.

Example 1

```
Input:
1

Output:
(3, 5)
```

Example 2

```
Input:
2

Output:
(5, 7)
```

Example 3

```
Input:
3

Output:
(11, 13)
```

Example 4

```
Input:
4

Output:
(17, 19)
```

July Fourth

Alexandra was born on fourth of July. She considers 4 and 7 to be her favorite digits. She really likes all the numbers that are made only of these two digits. For example: 4, 7, 477447, 77744774, 7474, 474, 4747474, 747, 44, 7777 - possibilities are endless. She calls these numbers *July Fourth Numbers*.

Recently she's been wondering about other interesting numbers. She is particularly interested in the numbers that can be evenly divided (i.e., without remainder) by her favorite numbers. She even came up with the name for these numbers: they are *July Fourth Family Numbers*.

Your job is to write a program that determines if a given number belongs to the *July Fourth Family Numbers*. Note that all *July Fourth Numbers* are a subset of *July Fourth Family Numbers* since each *July Fourth Number* can be divided evenly by itself.

Input

Input is a single line containing a number 1 <= n <= 1,000,000.

Output

A single line containing the string "July Fourth Family Number" or "Not in the family" - depending on what your program decides about the given input value.

Example 1

```
Input:
74

Output:
July Fourth Family Number
```

Example 2

```
Input:
8

Output:
July Fourth Family Number
```

Because 8 can be evenly divided by 4.

Example 3

```
Input:
47471

Output:
Not in the family
```

Chess Championship

In the World Chess Championship of 2035, 2^N contestants were supposed to participate. Because of sudden turn of the hurricane Alice, many flights were canceled last minute and many of the contestants could not arrive on time to participate in the championship.

With all the matches pre-arranged ahead of time, the organizer decide to keep the schedule as is and allow *walkover matches* in case when one of the players is missing.

- If both players are available, then there will be a normal match.
- If only one player is available, then this is a *walkover match* and that player automatically advances to the next level.
- If no player is available, then there is no match.

In the figure on the right, the players 3 and 4 could not arrive. Their match is canceled. Players 1 and 2, play their regular match and the winner advances to the next level match. But there is no second player there, so it is a *walkover match*.

Given the list of players who could not arrive on time, calculate the number of the *walkover matches* in the whole championship.

Input

Each test begins with two integers: $1 <= N <= 10$ and $0 <= M <= 2^N$. 2^N is the number of players scheduled for the championship. M is the number of players who could not arrive on time due to the canceled flights. The next line contains M integers, denoting the players who have not arrived. The players are numbered 1 to 2^N.

Output

The number of *walkover matches*.

Example 1

```
Input:
2 2
3 4
Output:
1
```

Example 2

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

Example 3

```
Input:
2 1
2
Output:
1
```

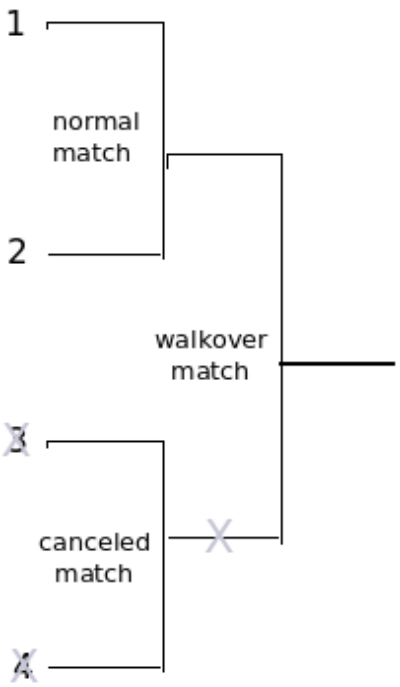

Figure 1: Illustration of example 1.

Bitly

As a new employer at Intel, you are faced with a notoriously repetitive job of manipulating register bits for configuring processors. The CS brain inside of you just wants an easier way to manipulate these bits with style and efficiency. You are generally faced with the following tasks in bit manipulation: setting and clearing a specific bit, and performing the logical OR/AND operation of any 2 bits. The register always initially contains all zeros.

To streamline your work, you are going to write a program that can *understand* the following commands:

- `ONE i` - puts a 1 at bit position i
- `ZERO i` - puts a 0 at bit position i
- `EITHER i j` - stores at i the logical OR operation result of bit i and bit j
- `BOTH i j` - stores at i the logical AND operation result of bit i and bit j

Since this is a 32-bit register, the values of i and j are always $0 <= i,j < 32$. Within the register the leftmost bit is the most significant bit (index 31) and the rightmost bit is the least significant bit (index 0).

Input

The first line of input contains the number of commands that follow, $0 < n <= 100$. The next n lines contain commands as indicated above each on a separate line.

Output

Content of the register by itself (no newline character at the end). The least significant bit should be on the right and the most significant bit should be on the left.

Example 1

```
Input:
2
ONE 20
EITHER 1 20

Output:
00000000000100000000000000000010
```

In the first operation bit position 20 was flipped to 1. In the second operation the result of logical OR between bits 20 and 1 is written to bit at position 1.

Example 2

```
Input:
4
BOTH 0 1
ONE 0
EITHER 3 0
ZERO 3

Output:
00000000000000000000000000000001
```

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
```