Bit-wise OR

Given 3 non-negative integers N, L, R. Find the **minimum** integer x such that

1. L <= x <= R
2. x **OR** N is maximized (**OR** stands for bit-wise OR operation here), i.e. for all integers y in the close interval [L,R], we have: (y **OR** N) <= (x **OR** N)

Input

There will be 3 integers N, L, R on a single line, where L <= R. You can assume that 0 <= N,L,R <= 4294967295.

Output

Print a line containing the value of x such that x **OR** N is maximized.

Example 1

```
Input:
10 10 10

Output:
10
```

Example 2

```
Input:
100 50 60

Output:
59
```

Efficient Adding

You are tasked with writing a program that adds a sequence of numbers. But the added challenge is to do so efficiently!

The cost of adding two numbers `a` and `b` is equal to their sum `a+b`. For example: to add 1, 2, and 3, you can do it as follows:

1 + 2 = 3, cost of 3 3 + 3 = 6, cost of 6 Total cost = 9

or

2 + 3 = 5, cost of 5 1 + 5 = 6, cost of 6 Total cost = 11

or

1 + 3 = 4, cost of 4 2 + 4 = 6, cost of 6 Total cost = 10

Your goal is to add the numbers so that the cost is as small as possible.

Input

The first line of input contains a positive number `N` (`2 <= N <= 5000`) that tells you how many numbers there are to add.

The second line of input contains those `N` numbers `0 <= n_1, n_2, ..., n_N <= 100,000`.

Output

The minimum total cost of addition followed by a newline.

Example 1

```
Input:
3
1 2 3

Output:
9
```

Example 2

```
Input:
4
1 2 3 4

Output:
19
```

Party Fee

Saya just graduated from her high school! She and her classmates rented a ballroom and held a party to celebrate their graduation. They realized that the room fee was not split evenly since some of them did not bring enough money. After going home, they decided to re-split the fee. However, the payment app they use only allows transfers between friends. As a kind person, you want to help them out.

You will be given information about how much money each of them owes or is owed and who their friends are. Your task is to figure out if it is possible for them to redistribute the party fee evenly. Remember that the money can only be transferred between friends and most students are friends only with a small subset of their classmates.

Input

The first line of the input contains two integers N ($2 <= N <= 10{,}000$) and M ($0 <= M <= 50{,}000$), representing the number of students and the number of friendships, respectively.

Each of the following N lines contains a single integer d ($-10{,}000 <= d <= 10{,}000$), indicating how much that student owes (if $d > 0$) or is owed (if $d < 0$). The student had already paid the right amount of the party fee if d is 0. It is guaranteed that sum of all d's is 0.

The following M lines describe friendships. Each of these M lines contains two integers a and b ($0 <= a < b < N$), meaning that the a-th and the b-th student are friends. Note that a friendship may appear more than once and it is also possible that someone is friends with themselves.

Output

You should print one line containing `POSSIBLE` or `IMPOSSIBLE`, indicating if it is possible for them to redistribute the party fee only by sending money to their friends. There is a new line character after the output string.

Example 1

```
Input:
5 3
100
-75
-25
-42
42
0 1
1 2
3 4

Output:
POSSIBLE
```

Example 2

```
Input:
4 2
15
20
-10
-25
0 2
1 3

Output:
IMPOSSIBLE
```

Move Union Find

In this problem you are to implement a variant of Union Find: *Move Union Find*.

The data structure you need to write is also a collection of disjoint sets, supporting 3 operations:

operation	description
`1 a b`	Union the sets containing `a` and `b`. If `a` and `b` are already in the same set, ignore this command
`2 a b`	Move the element `a` to the set containing `b`. If `a` and `b` are already in the same set, ignore this command
`3 a`	Return the number of elements and the sum of elements in the set containing `a`

Initially, the collection contains n sets: {1}, {2}, {3}, ..., {N}

Input

The first line contains two integer N and M. N is the total number of sets initially. M is the number of commands to follow. 1 <= N, M <= 100,000

Each of the next M lines contain a command. For each operation, 1 <= a,b <= N.

Output

For each type-3 command, output 2 integers: the number of elements and the sum of elements.

Example 1

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

Output:
3 7
```

Explanation:

Initial sets: {1}, {2}, {3}, {4}, {5}

After `1 1 2`: {1, 2}, {3}, {4}, {5}

After `2 3 4`: {1, 2}, {3, 4}, {5}

After `1 3 5}`: {1, 2}, {3, 4, 5}

After `2 4 1`: {1, 2, 4}, {3, 5}

Example 2

```
Input:
5 6
1 1 2
2 3 2
3 3
2 1 4
2 2 4
3 3
```

Output:
3 6
1 3

Line Up

Queuing, it's what the British are renowned for doing - and doing very well. Better than anyone else in the world, if reputation is to be believed. Today, queues can be found just about anywhere (even in the virtual world).

Standard *Queues* are well known to most computer science students. It's a first-in-first-out data structure. Kobayashi is already familiar with it. But she is assigned to write some code to maintain a variant of Standard Queues: **Group Queues**.

In a group queue each element belongs to a group. If an element is pushed to the queue and there is no element that belongs to the same group in the queue, it will be placed behind the last element in the queue. Otherwise, if there are already some elements that belong to the same group, it should be placed immediately behind them.

Dequeuing works the same as int the standard queue: We will remove the first element according to the current order from head to tail in the queue.

Kanna is preparing for her school's Sports Festival, and she wants Kobayashi to come to this important event. But Kobayashi will not be able to attend unless she can finish her program early. As Kobayashi's colleague and close friend, you decide to help her. You need to write a program that simulates such a group queue.

Input

The 1st line contains an integer `n` (`1 <= n <= 1,000`) equal to the number of groups, as described above.

Then `n` group descriptions follow. Each group is described on one line by an integer `k` (the number of elements belonging to the group) followed by `k` integers (the indexes of the elements in this group).

Elements are indexed with integer in the range `[ 0 , 999,999 ]`. And a group may consist of up to `1,000` elements.

Finally, a list of commands follows. There are 3 different kinds of commands:

- `Push ind` : Push the element with index `ind` into the group queue.
- `Pop` : Output the first element and remove it from the queue.
- `Shutdown` : Stop your program (end of input).

There may be up to `200,000` commands in the input file, so the implementation of the group queue should be efficient: both enqueuing and dequeuing of an element should only take a constant time.

Output

For each `Pop` command, print an integer, the index of the element which is dequeued, followed by a newline.

Example 1

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 4
Push 2
Push 5
Push 3
Push 6
Pop
Pop
Pop
Pop
Pop
```

```
Pop
Shutdown

Output:
1
2
3
4
5
6
```

Example 2

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 2
Push 4
Pop
Pop
Pop
Push 3
Push 5
Pop
Pop
Push 6
Pop
Pop
Shutdown

Output:
1
2
4
5
3
6
```