Alice, Bob, and Circuit

The Cyberland Circuit Foundation consists of n members. Each member has his/her favorite number and a unique name (the favorite numbers may not be distinct).

m letters have been sent between the members. Each letter has a sender and a recipient, and the content of the letter is the sender's favorite number.

Each member calculates the sum of the contents (senders' favorite numbers) they received and takes the modulo of 65536 (i.e., 2^{16}) as his/her result number.

Your task is to determine all result numbers.

However, the situation is not as straightforward as it seems. Alice, Bob, and Circuit decide to solve this problem in a slightly more complicated way:

- Alice knows all n members (name and favorite number), but knows no information about letters. She needs to send a binary string to Circuit with a length of no more than 10^5.
- Bob knows all m letters (sender and recipient's name), but knows no information about members. He needs to send a binary string to Circuit with a length of no more than 10^5.
- Circuit can receive binary strings sent by Alice and Bob, and subsequently generate a binary string comprising $16n$ bits as output. However, due to its limited computational power, Circuit is only capable of performing basic logical operations (e.g., AND, OR, NOT).

In the following, we will introduce how the circuit works in detail.

Circuit Details

The gate is the basic element of a circuit. A gate consists of zero or two boolean inputs(depending on the type of gate), and one boolean output. There are two types of gates: input gates and computation gates.

- Input gates have no input and represent the bits from binary strings sent by Alice and Bob.
  - There will be $l_A + l_B$ input gates, labeled from 0 to $(l_A + l_B - 1)$, where l_A, l_B are the lengths of the strings from Alice and Bob, respectively.
  - For $0 \le i < l_A$, the output of i-th gate is the i-th bit of the string from Alice;
  - For $0 \le i < l_B$, the output of $(i + l_A)$-th gate is the i-th bit of the string from Bob.
- Computation gates have two inputs and represent the computation process.
  - The labels for computation gates start from $(l_A + l_B)$.

- For each computation gate, you should provide labels of two dependent gates for input, and the operation type $p(0 \le p \le 15)$.
  - To prevent circular dependencies, the labels of the two dependent gates must be smaller than the label of the computation gate.
  - If outputs of its two dependent gates are x_0 and x_1 respectively ($x_0, x_1 \in \{0,1\}$), then the output of the computation gate is:

$$f(p, x_0, x_1) = \left\lfloor \frac{p}{2^{x_0+2x_1}} \right\rfloor \bmod 2$$

Here are some examples that may be useful for you:

x_0	x_1	x_0 **AND** x_1 $f(8, x_0, x_1)$	x_0 **OR** x_1 $f(14, x_0, x_1)$	x_0 **XOR** x_1 $f(6, x_0, x_1)$	**NOT** x_0 $f(5, x_0, x_1)$
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Implementation Details

Please note:

- All array indices start from 0. For example, if `a` is an array of length `n`, then `a[0]` to `a[n-1]` are valid data, accessing indices beyond that range may cause an out-of-bounds error.
- All strings are terminated by a null character `\0`.

You should implement the following procedures:

Alice

```
int alice(const int n, const char names[][5],
          const unsigned short numbers[], bool outputs_alice[]);
```

Direction	Value	Length	Meaning	Constraint
Input	n	1	n	$0 \le n \le 700$
	names	n	The name of each member.	All names are distinct, consisting of lowercase English letters only, and have a maximum length of 4 characters.

<table>
<tr><th>Direction</th><th>Value</th><th>Length</th><th>Meaning</th><th>Constraint</th></tr>
<tr><td></td><td>numbers</td><td>n</td><td>The favorite number of each member.</td><td>Each number is in the range from 0 to 65535.</td></tr>
<tr><td rowspan="2">Output</td><td>outputs_alice</td><td>l_A</td><td>The binary string is sent to Circuit.</td><td></td></tr>
<tr><td>(Return value)</td><td>1</td><td>l_A</td><td>You need to make sure that l_A does not exceed 10^5 and when n is the same, l_A must be fixed.</td></tr>
</table>

Bob

```
int bob(const int m, const char senders[][5],
        const char recipients[][5], bool outputs_bob[]);
```

<table>
<tr><th>Direction</th><th>Value</th><th>Length</th><th>Meaning</th><th>Constraint</th></tr>
<tr><td rowspan="3">Input</td><td>m</td><td>1</td><td>m</td><td>$0 \le m \le 1000$</td></tr>
<tr><td>senders</td><td>m</td><td>The sender's name on each letter.</td><td rowspan="2">All names appear in Alice's input</td></tr>
<tr><td>recipients</td><td>m</td><td>The recipient's name on each letter.</td></tr>
<tr><td rowspan="2">Output</td><td>outputs_bob</td><td>l_B</td><td>The binary string is sent to Circuit.</td><td></td></tr>
<tr><td>(Return value)</td><td>1</td><td>l_B</td><td>You need to make sure that l_B does not exceed 10^5 and when m is the same, l_B must be fixed.</td></tr>
</table>

Circuit

To ensure that the computation process of the Circuit is like a general circuit, you cannot directly obtain the binary strings sent from Alice and Bob to Circuit. You only know the lengths of these two strings and output the circuit structure.

```
int circuit(const int la, const int lb, int operations[],
            int operands[][2], int outputs_circuit[][16]);
```

Direction	Value	Length	Meaning	Constraint
Input	la	1	l_A	
	lb	1	l_B	
Output	operations	l	The type of operation performed by each gate in the circuit.	An integer from 0 to 15.
	operands	l	The operand used by each gate in the circuit.	The number must be less than the label of the current gate.
	outputs_circuit	n	The gate label of the circuit output.	`outputs_circuit[i][j]` denotes the j-th bit (counting from the least significant bit) of the final result for the i-th member. The members are ordered according to Alice's input.
	(Return value)	1	l, which represents the total number of gates (including input gates).	**You need to ensure that $l \le 2 \times 10^7$**

Although you can modify the information of gates with indices less than $l_A + l_B$ in the `operations` and `operands` arrays, the grader would ignore such modification.

Example

Consider the following calls:

```
alice(3, {"alic", "bob", "circ"}, {10000, 20000, 30000}, outputs_alice);
bob(5, {"alic", "bob", "bob", "circ", "circ"},
    {"circ", "circ", "alic", "circ", "circ"}, outputs_bob);
```

It represents the following scenario:

- Alice knows there are 3 members, the member with the name `alic` has a favorite number 10000, etc. A possible output for `alice()` is that,
  - The return value of `alice()` is 2, representing $l_A = 2$.
  - Inside `alice()` function, set `outputs_alice[0] = 1`, `outputs_alice[1] = 0`, representing that the result binary string is 10.
- Bob knows there are 5 letters, the first letter is from `alic` to `circ`, etc. A possible output for `bob()` is that,
  - The return value of `bob()` is 3, representing $l_B = 3$.
  - Inside `bob()` function, set `outputs_bob[0] = 1`, `outputs_bob[1] = 1`, `outputs_bob[2] = 0`, representing that the result binary string is 110.

Based on previous outputs for `alice()` and `bob()`, there will be the following call:

```
circuit(2, 3, operations, operands, outputs_circuit);
```

A correct output for this function would be

- The return value of `circuit()` is 7, meaning that we add two computation gates, labeled 5 and 6.
- Inside `circuit()`, set `operations`, `operands`, and `outputs_circuit` in the following way:
  - `operations = {-1, -1, -1, -1, -1, 8, 14}`, where we use `-1` to represent ignored information from input gates;
  - `operands = {{-1, -1}, {-1, -1}, {-1, -1}, {-1, -1}, {-1, -1}, {0, 4}, {2, 5}};`
  - `outputs_circuit = {{5, 5, 5, 5, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 6, 5}, ...}`. The array is a bit long, you can check `abc.cpp` in the attachments for the full array.

According to the output, the computation procedure is that,

- Add a type 8 computation gate, with input from gate 0 and gate 4. The output of gate 0 is the 0-th bit of the string from Alice, which is 1; The output of gate 4 is the 2-nd bit of the string from Bob, which is 0. So the output for gate 5 is $f(8, 0, 1) = 0 \text{ AND } 1 = 0$.
- Add a type 14 computation gate, with input from gate 2 and gate 5. The output of gate 2 is the 0-th bit of the string from Bob, which is 1; The output of gate 5 is 0. So the output for gate 6 is $f(14, 1, 0) = 1 \text{ OR } 0 = 1$.
- `output_circuit[0]` represents the final result of `alic`, which is $(0100111000100000)_2 = 20000$. Since `alic` only receives a letter from `bob`, the final result of `alic` is 20000.
- The final result of `bob` should be 0, since he receives no letter; The final result of `circ` should be $(10000 + 20000 + 30000 + 30000) \bmod 65536 = 24464$.

`abc.cpp` in the attachments can pass this example, but we do not guarantee that it can pass other test cases.

Constraints

For all test cases:

- $0 \leq n \leq 700, 0 \leq m \leq 1000$.
- All names are distinct, consisting of lowercase English letters only, and have a maximum length of 4 characters.
- The favorite number of each member is in the range of 0 to 65535.
- The names of all senders and recipients appear in Alice's input array `names`.
- `alice()` and `bob()` have a memory limit of 2048 MiB and a time limit of 0.02 seconds, respectively.
- `circuit()` has a memory limit of 2048 MiB and a time limit of 7 seconds.

For the final evaluation, `alice()` and `bob()` may be called multiple times in a single test case. The time limit of 0.02 second is for each call.

Subtasks

Subtask Type A (12 points)

Subtask 1,2,3 are in subtask type A, where $n = 1$.

Each subtask has the following additional constraints:

- Subtask 1 (4 points): $m = 0$.
- Subtask 2 (4 points): $0 \leq m \leq 1$.
- Subtask 3 (4 points): $0 \leq m \leq 1000$.

Subtask Type B (54 points)

Subtask 4,5,6 are in subtask type B, where:

- $0 \leq n \leq 30, \frac{n}{2} \leq m \leq n^2$.
- There are no two letters with the same sender and recipient.
- All member names appear in Bob's input (i.e., each member either sends at least one letter or receives at least one letter).

Each subtask has the following additional constraints:

- Subtask 4 (24 points): $n = 26$, All members' names are single lowercase letters, and in Alice's input, they appear in order from `a` to `z`.
- Subtask 5 (24 points): $n = 26$.
- Subtask 6 (6 points): No special restrictions.

Subtask Type C (34 points)

Subtask 7,8,9 are in subtask type C, where $0 \le n \le 700, 0 \le m \le 1000$.

Each subtask has the following additional constraints:

- Subtask 7 (18 points): $n = 676$, all members' names are two lowercase letters, and in Alice's input, they appear in lexicographical order (e.g., `aa`, `ab`, `ac`, ..., `az`, `ba`, ..., `bz`, `ca`, ..., `zz`).
- Subtask 8 (10 points): $n = 676$.
- Subtask 9 (6 points): No additional constraints.

Sample Grader

The sample grader reads the input in the following format:

- Line 1: $n\ m$
- Line $2 + i (0 \le i \le n - 1)$: $names_i\ numbers_i$
- Line $2 + n + i (0 \le i \le m - 1)$: $senders_i\ recipients_i$.

The sample grader outputs in the following format:

- If the program finishes successfully, the sample grader will output n lines, each containing an integer, representing the final result calculated by functions you implement for each member.
- Otherwise, the grader would output nothing to stdout and prints the error messages to the file `abc.log` in the directory.
- Additionally, the sample grader will output values of l_A, l_B, l and the running time of each function to `abc.log`.

The sample grader will not check the memory limit and the restriction that for the same n / m, l_A / l_B must be equal.