# Fast Hash Transform

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 5 seconds |
| Memory limit: | 512 megabytes |

Define a closed function $f$ on $\mathcal{V} = \left\{0, 1 \cdots, 2^{64} - 1\right\}$ as a single-round hash function if and only if it has the following standard form:

$$f(X) = B \oplus \bigoplus_{j=1}^{m} \left( (X \lll s_j) \circ_j A_j \right),$$

where,

- $A_j, B \in \mathcal{V}$;
- $0 \leq m \leq 64$, $0 \leq s_j \leq 63$ and all $s_j$ are distinct;
  - $\oplus$ denotes bitwise XOR operation, $\lll$ denotes circular left shift operation (the circular left shift on $\mathcal{V}$ is defined as circular left shifting the unsigned binary representation padded to 64 bits), and $\circ_j$ denotes either the bitwise OR operation (denoted as $\vee$) or the bitwise AND operation (denoted as $\wedge$).

Given $N$ single-round hash functions $f_1, f_2, \cdots, f_N$ and $Q$ operations, the operations are of the following two types:

- Given $l, r, x$, compute the value of $f_r\left(\cdots\left(f_{l+1}\left(f_l(x)\right)\right)\cdots\right)$;
- Given $l$, modify $f_l$ to a new single-round hash function.

Considering the practical aspect that modifying single-round hash functions would invalidate a large number of quick hash transformation results, it is guaranteed that such modifications will not exceed $C$ times.

## Input

The first line contains three non-negative integers $N, Q, C$, representing the number of single-round hash functions to be maintained, the number of operations, and the number of operations to modify a single-round hash function, respectively. It is guaranteed that $1 \leq N, Q \leq 20,000$, $0 \leq C \leq 400$, and $C < Q$.

From the second line to the $(N+1)$-th line, the $(i+1)$-th line describes the initial parameters of $f_i$:

- The first non-negative integer $m$ in each line indicates the number of terms involved in the XOR sum in the standard form of $f_i$. It is guaranteed that $0 \leq m \leq 64$.
- Next, input $3m$ non-negative integers in sequence, where the $(3j-2)$-th, $(3j-1)$-th, and $3j$-th integers are $s_j, o_j, A_j$, respectively, representing the number of bits $X$ is circularly left-shifted in the $j$-th term of the XOR sum, the type of bitwise operation $\circ_j$ in the $j$-th term ($o_j = 0$ corresponds to bitwise OR operation, $o_j = 1$ corresponds to bitwise AND operation), and the constant term of the bitwise operation in the $j$-th term. It is guaranteed that $0 \leq s_j \leq 63$ and the $s_j$ values are distinct within the same single-round hash function, $0 \leq o_j \leq 1$, $0 \leq A_j < 2^{64}$.
- The last number in each line is a non-negative integer $B$, representing the constant term in the outer layer of the standard form. It is guaranteed that $0 \leq B < 2^{64}$.

From the $(N+1)$-th to the $(N+Q+1)$-th line, the $(N+i+1)$-th line describes the $i$-th operation.

- The beginning of each line is a non-negative integer $op$, indicating the type of operation.
- If $op = 0$, it indicates an operation to compute a quick hash transformation. It is followed by three non-negative integers $l, r, x$, indicating the starting position, ending position, and initial value of the quick hash transformation, respectively. It is guaranteed that $1 \le l \le r \le N$, $0 \le x < 2^{64}$.
- If $op = 1$, it indicates an operation to modify a certain single-round hash function. First, input an integer $l$, indicating the number of the single-round hash function to be modified. Then, input several integers in sequence to describe the modified $f_l$. The format for describing the new single-round hash function is the same as the format for describing the initial parameters of each function. It is guaranteed that $1 \le l \le N$.

## Output

For each of the operation of the first type, output a single line contains a single integer, indicating the answer.

## Example

| standard input | standard output |
|---|---|
| 3 5 1<br>1 4 0 0 51966<br>1 60 0 0 0<br>1 0 0 16 15<br>0 1 1 771<br>0 2 2 32368<br>0 3 3 0<br>1 2 2 0 0 15 61 1 4095 46681<br>0 1 3 2023 | 64206<br>2023<br>31<br>1112 |

## Note

For the first example:

The initial parameters for the 3 single-round hash functions are:

- $f_1(X) = (X \lll 4) \oplus 51996$ (where 51996 is represented in hexadecimal as '0xCAFE');
- $f_2(X) = X \lll 60$;
- $f_3(X) = (X \vee 16) \oplus 15$ (where 16 and 15 are represented in hexadecimal as '0x0010' and '0x000F' respectively).

The first operation is to compute a quick hash transformation. The result of performing a quick hash transformation on 771 ('0x0303') is $f_1(771) = (771 \lll 4) \oplus 51996 = 64206$ ('0xFACE').

The second operation is to compute a quick hash transformation. The result of performing a quick hash transformation on 32368 ('0x7E70') is $f_2(32368) = 32368 \lll 60 = 2023$ ('0x07E7').

The third operation is to compute a quick hash transformation. The result of performing a quick hash transformation on 0 ('0x0000') is $f_3(0) = (0 \vee 16) \oplus 15 = 31$ ('0x001F').

The fourth operation is to modify $f_2$. After modification, $f_2(X) = (X \vee 15) \oplus ((X \lll 61) \wedge 4095) \oplus 46681$ (where 4095 and 46681 are represented in hexadecimal as '0x0FFF' and '0xB659' respectively).

The fifth operation is to compute a quick hash transformation. For the initial value 2023 (‘0x07E7‘), since $f_1(2023) = 46222$ (‘0xB48E‘), $f_2(46222) = 1095$ (‘0x0447‘), and $f_3(1095) = 1112$ (‘0x0458‘), the result of the quick hash transformation is 1112.

Please note the impact of input and output efficiency on program running time.

In the example explanation, all hexadecimal representations are padded to 4 hexadecimal digits, but this does **not** mean that the input numbers do not exceed 65535.