IOI '24 P2 - Message

Time limit: 3.0s **Memory limit:** 1G

Aisha and Basma are two friends who correspond with each other. Aisha has a message M, which is a sequence of S bits (i.e., zeroes or ones), that she would like to send to Basma. Aisha communicates with Basma by sending her **packets**. A packet is a sequence of 31 bits indexed from 0 to 30. Aisha would like to send the message M to Basma by sending her some number of packets.

Unfortunately, Cleopatra compromised the communication between Aisha and Basma and is able to **taint** the packets. That is, in each packet Cleopatra can modify bits on exactly 15 indices. Specifically, there is an array C of length 31, in which every element is either 0 or 1, with the following meaning:

- $C[i] = 1$ indicates that the bit with index i can be changed by Cleopatra. We call these indices **controlled** by Cleopatra.
- $C[i] = 0$ indicates that bit with index i cannot be changed by Cleopatra.

The array C contains precisely 15 ones and 16 zeroes. While sending the message M, the set of indices controlled by Cleopatra stays the same for all packets. Aisha knows precisely which 15 indices are controlled by Cleopatra. Basma only knows that 15 indices are controlled by Cleopatra, but she does not know which indices.

Let A be a packet that Aisha decides to send (which we call the **original packet**). Let B be the packet that is received by Basma (which we call the **tainted packet**). For each i, such that $0 \leq i < 31$:

- if Cleopatra does not control the bit with index i ($C[i] = 0$), Basma receives bit i as sent by Aisha ($B[i] = A[i]$),
- otherwise, if Cleopatra controls the bit with index i ($C[i] = 1$), the value of $B[i]$ is decided by Cleopatra.

Immediately after sending each packet, Aisha learns what the corresponding tainted packet is.

After Aisha sends all the packets, Basma receives all the tainted packets **in the order they were sent** and has to reconstruct the original message M.

Your task is to devise and implement a strategy that would allow Aisha to send the message M to Basma, so that Basma can recover M from the tainted packets. Specifically, you should implement two procedures. The first procedure performs the actions of Aisha. It is given a message M and the array C, and should send some packets to transfer the message to Basma. The second procedure performs the actions of Basma. It is given the tainted packets and should recover the original message M.

Implementation Details

The first procedure you should implement is:

```
void send_message(std::vector<bool> M, std::vector<bool> C)
```

- M: an array of length S describing the message that Aisha wants to send to Basma.
- C: an array of length 31 indicating the indices of bits controlled by Cleopatra.

- This procedure may be called **at most 2100 times** in each test case.

This procedure should call the following procedure to send a packet:

```
std::vector<bool> send_packet(std::vector<bool> A)
```

- A: an original packet (an array of length 31) representing the bits sent by Aisha.
- This procedure returns a tainted packet B representing the bits that will be received by Basma.
- This procedure can be called at most 100 times in each invocation of `send_message`.

The second procedure you should implement is:

```
std::vector<bool> receive_message(std::vector<std::vector<bool>> R)
```

- R: an array describing the tainted packets. The packets originate from packets sent by Aisha in one `send_message` call and are given **in the order they were sent** by Aisha. Each element of R is an array of length 31, representing a tainted packet.
- This procedure should return an array of S bits that is equal to the original message M.
- This procedure may be called **multiple times** in each test case, **exactly once** for each corresponding `send_message` call. The **order of** `receive_message` **procedure calls** is not necessarily the same as the order of the corresponding `send_message` calls.

Note that in the grading system the `send_message` and `receive_message` procedures are called in **two separate programs**.

Constraints

- $1 \leq S \leq 1024$
- C has exactly 31 elements, out of which 16 are equal to 0 and 15 are equal to 1.

Subtasks and Scoring

If in any of the test cases, the calls to the procedure `send_packet` do not conform to the rules mentioned above, or the return value of any of the calls to procedure `receive_message` is incorrect, the score of your solution for that test case will be 0.

Otherwise, let Q be the maximum number of calls to the procedure `send_packet` among all invocations of `send_message` over all test cases. Also let X be equal to:

- 1, if $Q \leq 66$
- 0.95^{Q-66}, if $66 < Q \leq 100$

Then, the score is calculated as follows:

Subtask	Score	Additional Constraints
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	No additional constraints.

Note that in some cases the behaviour of the grader can be **adaptive**. This means that the values returned by `send_packet` may depend not just on its input arguments but also on many other things, including the inputs and return values of the prior calls to this procedure and pseudo-random numbers generated by the grader. The grader is **deterministic** in the sense that if you run it twice and in both runs you send the same packets, it will make the same changes to them.

Example

Consider the following call.

```
send_message([0, 1, 1, 0],
             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

The message that Aisha tries to send to Basma is $[0, 1, 1, 0]$. The bits with indices from 0 to 15 cannot be changed by Cleopatra, while the bits with indices from 16 to 30 can be changed by Cleopatra.

For the sake of this example, let us assume that Cleopatra fills consecutive bits she controls with alternating 0 and 1, i.e. she assigns 0 to the first index she controls (index 16 in our case), 1 to the second index she controls (index 17), 0 to the third index she controls (index 18), and so on.

Aisha can decide to send two bits from the original message in one packet as follows: she will send the first bit at the first 8 indices she controls and the second bit at the following 8 indices she controls.

Aisha then chooses to send the following packet:

```
send_packet([0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Note that Cleopatra can change bits with the last 15 indices, so Aisha can set them arbitrarily, as they might be overwritten. With the assumed strategy of Cleopatra, the procedure returns:
$[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]$.

Aisha decides to send the last two bits of M in the second packet in a similar way as before:

```
send_packet([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

With the assumed strategy of Cleopatra, the procedure returns:
$[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]$.

Aisha can send more packets, but she chooses not to.

The grader then makes the following procedure call:

```
receive_message([[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
                 [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]])
```

Basma recovers message M as follows. From each packet she takes the first bit that occurs twice in a row, and the last bit that occurs twice in a row. That is, from the first packet, she takes bits $[0, 1]$, and from the second packet she takes bits $[1, 0]$. By putting them together, she recovers the message $[0, 1, 1, 0]$, which is the correct return value for this call to `receive_message`.

It can be shown that with the assumed strategy of Cleopatra and for messages of length 4, this approach of Basma correctly recovers M, regardless of the value of C. However, it is not correct in the general case.

Sample Grader

The sample grader is not adaptive. Instead, Cleopatra fills consecutive bits she controls with alternating 0 and 1 bits, as described in the example above.

Input format: **The first line of the input contains an integer T, specifying the number of scenarios.** T scenarios follow. Each of them is provided in the following format:

```
S
M[0]  M[1]  ...  M[S-1]
C[0]  C[1]  ...  C[30]
```

Output format: The sample grader writes the result of each of the T scenarios in the same order as they are provided in the input in the following format:

```
K L
D[0]  D[1]  ...  D[L-1]
```

Here, K is the number of calls to `send_packet`, D is the message returned by `receive_message` and L is its length.

Time and Memory Limit Computation

A submission's displayed execution time is calculated as the **sum** of the execution times of both programs. However, a submission will be accepted if **both** programs execute under the time limit: for example, if the first program executes in `2.5s` and the second program executes in `1.6s`, the submission will not TLE, and the time displayed will be `4.1s`.

A submission's memory usage is the **maximum** of the memory usages of both programs.

Attachment Package

The sample grader along with sample test cases are available here.