메시지

Aisha와 Basma는 서로 연락하는 친구이다. Aisha는 메시지 M을 가지고 있는데 이 메시지는 S 개의 0과 1로 구성된 비트열이다. 그녀는 이 메시지를 Basma에게 보내려고 한다. Aisha는 Basma에게 이 메시지를 여러개의 **패킷**으로 바꿔서 전송하려고 한다. 한 패킷은 31개의 0과 1로 이루어진 비트열이며 각 비트는 0번부터 30번까지로 인덱싱된다.

불행하게도, Cleopatra는 Aisha와 Basma의 통신을 방해하고 패킷들을 **오염**시킬수 있다. 좀 더 정확하게 말하면, Cleopatra는 한 패킷에서 정확하게 15 개 인덱스의 비트를 변경할 수 있다. 31 개의 비트로 구성된 배열 C는 다음과 같은 의미를 가진다.

- $C[i] = 1$은 Cleopatra가 i번째 인덱스 비트를 변경할 수 있다는 의미이다. 우리는 Cleopatra가 이러한 비트를 **통제**한다고 부르겠다.
- $C[i] = 0$은 Cleopatra가 i번째 인덱스 비트를 변경할 수 없다는 의미이다.

배열 C는 15 개의 1과 16 개의 0으로 구성되어 있다. 메시지 M을 전송하는 동안 Cleopatra가 통제하는 비트의 인덱스는 모든 패킷에 대해서 동일하다. Aisha는 Cleopatra가 통제하는 15 개 비트의 인덱스를 정확하게 알고 있다. Basma는 Cleopatra가 통제하는 비트의 수가 15 개라는 것은 알지만 통제되는 비트의 정확한 위치는 알지 못한다.

Aisha가 보내려고 하는 패킷을 A라고 하자. (A는 **원본 패킷**이라고 부르겠다.) Basma가 받는 패킷은 B라고 하자. (B는 **오염된 패킷**으로 부르겠다.) 각 인덱스 i에 대해서 ($0 \leq i < 31$) 다음을 만족한다.

- Cleopatra가 i번째 인덱스 비트를 통제하지 못하면 ($C[i] = 0$), Basma는 Aisha가 보낸 i번째 인덱스 비트를 받는다. ($B[i] = A[i]$),
- 반대로 Cleopatra가 i번째 인덱스 비트를 통제하면 ($C[i] = 1$), $B[i]$ 값은 Cleopatra가 결정한다.

Aisha는 원본 패킷 A를 전송한 직후에 오염된 패킷 B가 무엇인지 알 수 있다. Aisha가 모든 원본 패킷 A들을 전송한 다음에, Basma는 모든 오염된 패킷 B들을 **전송된 순서**대로 받는다. 이후, Basma는 원본 메시지 M을 복원해야 한다.

여러분은 Aisha가 메시지 M을 전송해서 Basma가 오염된 패킷에서 메시지 M을 복원하는 전략을 고안해서 구현하여야 한다. 좀더 구체적으로 말하면, 여러분은 2 개의 프로시져를 구현해야 한다. 첫번째 프로시져는 Aisha의 행동을 구현한다. 이 프로시져는 메시지 M과 배열 C를 받아서 메시지를 전달할 수 있는 패킷 A들을 전송해야 한다. 두번째 프로시져는 Basma의 행동을 구현한다. 이 프로시져는 오염된 패킷 B들을 받아서 원본 메시지 M을 복원해야 한다.

Implementation Details

여러분이 구현할 첫번째 프로시져는 다음과 같다:

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

- M: 길이가 S인 배열로 Aisha가 Basma에게 전달하려는 메시지
- C: 길이가 31인 배열로 Cleopatra가 통제하는 비트의 인덱스를 표시하는 배열
- 이 프로시져는 한 테스트케이스에서 **최대 2100 번까지** 호출될 수 있다.

이 프로시져는 한 개의 패킷을 전송하기 위해서 다음의 프로시져를 호출해야 한다:

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

- A: Aisha가 전송한 비트를 표현하는 원본 패킷 (길이가 31인 배열)
- 이 프로시져는 Basma가 받을 비트를 표현하는 오염된 패킷 B를 반환한다.
- 한 번의 send_message 호출에서 이 프로시져를 최대 100번까지 호출할 수 있다.

여러분이 구현할 두번째 프로시져는 다음과 같다:

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

- R: 오염된 패킷을 나타내는 배열. Aisha가 보낸 패킷으로 send_message를 호출해서 만들어진 패킷들이 Aisha가 전송한 순서대로 주어진다. R 배열의 각 원소는 길이가 31인 배열이며 한 개의 오염된 패킷을 나타낸다.
- 이 프로시져는 S 비트 배열을 반환해야 하며 이는 원본 메시지 M과 동일해야 한다.
- 이 프로시져는 한 테스트 케이스에서 **여러번** 호출될 수 있고 한 번의 send_message 호출에 대해서는 **단 한번만** 호출된다. receive_message **프로시져 호출들**의 **순서**는 관련된 send_message 호출의 순서와 다를 수 있다.

채점시스템에서는 send_message와 receive_message 프로시져가 **두 개의 별도 프로그램**에서 호출된다.

제약조건

- $1 \leq S \leq 1024$
- C는 정확하게 31개의 원소를 가지며 그 중의 16개는 0이고 15개는 1이다.

서브태스크와 점수

테스트케이스 중에 하나라도 send_packet 프로시져에 대한 호출이 위에서 설명한 규칙과 맞지 않거나 receive_message 프로시져 호출의 반환값이 틀리면 해당 테스트케이스의 점수는 0점이 될 것이다.

그렇지 않으면 모든 테스트케이스의 send_message 프로시져의 모든 호출 중에 프로시져 send_packet의 호출 횟수의 최댓값을 Q라 하자.

또한 X는 다음과 같다:

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

그러면, 점수는 다음과 같이 계산된다.

서브태스크	점수	추가제약조건
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	추가제약조건 없음

어떤 경우에는 그레이더의 행동이 **적응적(adaptive)**임을 유의하라. 이 뜻은 send_packet 의 반환값이 입력 인자값 뿐만 아니라 이 프로시져의 이전 호출의 입력 인자값과 반환값 그리고 그레이더가 생성한 pseudo-random numbers 등 여러 요소에 의존할 수 있다는 뜻이다. 그레이더는 어떤 측면에서는 **결정적(deterministic)**인데 그 이유는 여러분이 그레이더를 두 번 실행하여 두 번 모두 같은 패킷들을 생성하면, 그레이더가 두 번 모두 동일한 변경을 수행하기 때문이다.

예제

다음 호출을 살펴보자.

```
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])
```

Aisha가 Basma에게 보내려는 메시지는 $[0, 1, 1, 0]$이다. Cleopatra는 16 번부터 30 번까지 인덱스의 비트는 변경할 수 있지만 0 번부터 15 번 인덱스의 비트는 변경할 수 없다.

이 예제에는 Cleopatra가 통제할 수 있는 연속적인 비트들에 대해 0과 1을 번갈아서 채운다고 가정하자. 즉, 그녀는 그녀가 통제하는 첫 번째 인덱스 (이 경우에는 인덱스 16) 비트를 0으로 채우고 두 번째 인덱스 (인덱스 17) 비트는 1로 채우고 세 번째 인덱스 (인덱스 18) 비트는 0으로 채우는 식이다.

Aisha는 다음과 같이 한 패킷에 원본 메시지의 두 비트씩 보낼 수 있다: 그녀는 처음 비트를 처음 8 개의 인덱스에 보내고 두번째 비트를 그 다음 8 개 인덱스에 보낼 것이다.

Aisha는 다음의 패킷을 보내기로 선택한다:

```
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])
```

Cleopatra는 마지막 15 개의 인덱스 비트를 변경할 수 있으므로 Aisha는 어차피 변경될 그 값들을 임의로 정해도 된다. 위에서 가정된 Cleopatra의 전략에 의하면 이 프로시져는 다음을 반환한다: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha는 M의 나머지 2 개의 비트를 두번째 패킷에 비슷한 방법으로 보내기로 한다:

```
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])
```

위에서 가정된 Cleopatra의 전략에 의하면 이 프로시져는 다음을 반환한다: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha는 더 많은 패킷을 보낼 수 있지만 안 보내기로 한다.

그레이더는 다음과 같이 프로시져를 호출한다:

```
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]])
```

Basma는 메시지 M을 다음과 같이 복원한다.각 패킷에서 두개가 연속으로 나오는 첫 비트를 선택하고 두 개가 연속으로 나오는 마지막 비트를 선택한다. 즉, 첫 패킷에서는 $[0,1]$을 선택하고 두번째 패킷에서는 $[1,0]$를 선택한다. 이 두개를 합쳐서 메시지 $[0,1,1,0]$를 만들 수 있는데 이는 `receive_message` 호출의 정확한 반환값이다.

Cleopatra가 가정한 전략을 사용할 때 길이가 4 인 메시지들에 대해서는 이 방법을 사용하면 Basma는 C의 값에 상관없이 항상 M을 복원할 수 있음을 보일 수 있다. 하지만 이 방법은 일반적인 방법은 아니다.

Sample Grader

sample grader는 적응적이지 않다. 대신에, Cleopatra의 행동은 위의 예제에서 설명한 대로 그녀가 통제할 수 있는 연속적인 비트를 번갈아가면서 0과 1로 채운다.

입력 포맷: **입력에 첫번째 줄에는 시나리오의 갯수 T가 주어진다.** T개 시나리오가 그 다음에 주어진다. 각각의 시나리오는 다음의 포맷으로 주어진다:

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

출력 포맷: sample grader는 각 T개의 시나리오의 결과를 입력에 주어진 순서대로 다음과 같이 출력한다:

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

여기서, K는 `send_packet`의 호출 횟수이고, D는 `receive_message`가 반환한 메시지이며 L은 그 길이이다.