Mensaje

Aisha y Basma son dos amigos que suelen enviarse mensajes. Aisha tiene un mensaje M, que es una secuencia de S bits (es decir, ceros o unos) que le gustaría enviar a Basma. Aisha se comunica con Basma enviándole **paquetes**. Un paquete es una secuencia de 31 bits indexados desde 0 hasta 30. A Aisha le gustaría enviar el mensaje M a Basma enviándole un cierto número de paquetes.

Desafortunadamente, Cleopatra comprometió la comunicación entre Aisha y Basma y puede **alterar** los paquetes. Específicamente, en cada paquete Cleopatra puede modificar los bits en exactamente 15 índices. Formalmente, hay un arreglo C de tamaño 31, en el que cada elemento es 0 o 1, con el siguiente significado:

- $C[i] = 1$ indica que el bit con índice i puede ser cambiado por Cleopatra. Llamaremos a estos índices los **controlados** por Cleopatra.
- $C[i] = 0$ indica que el bit con índice i no puede ser cambiado por Cleopatra.

El arreglo C contiene exactamente 15 unos y 16 ceros. Mientras se envía el mensaje M, el conjunto de índices controlados por Cleopatra se mantiene igual para todos los paquetes. Aisha sabe exactamente cuáles son los 15 índices controlados por Cleopatra. Basma solo sabe que hay 15 índices que están siendo controlados por Cleopatra pero ella no sabe cuáles son esos índices.

Sea A el paquete que Aisha decide enviar (al que llamaremos el **paquete original**). Sea B el paquete que es recibido por Basma (al que llamaremos el **paquete alterado**).

Para cada i, tal que $0 \leq i < 31$:

- si Cleopatra no controla el bit con índice i ($C[i] = 0$), Basma recibe el bit i igual a como lo envía Aisha ($B[i] = A[i]$).
- de lo contrario, si Cleopatra controla el bit con índice i ($C[i] = 1$), el valor del bit $B[i]$ es decidido por Cleopatra.

Inmediatamente después de enviar cada paquete, Aisha conoce cuál es el paquete alterado correspondiente.

Después de que Aisha envie todos los paquetes, Basma recibe todos los paquetes alterados **en el orden en que fueron enviados** y tiene que reconstruir el mensaje original M.

Tu tarea es elaborar e implementar una estrategia que permita a Aisha mandar el mensaje M a Basma, tal que Basma pueda reconstruir el mensaje M a partir de los paquetes alterados. Específicamente, tienes que implementar las siguientes dos funciones. La primera función ejecuta

las acciones de Aisha: Toma como entrada un mensaje M y el arreglo C, y debe enviar algunos paquetes para enviar el mensaje a Basma. La segunda función ejecuta las acciones de Basma: Toma como entrada los paquetes alterados y tiene que reconstruir el mensaje original M.

Detalles de Implementación

La primera función que debes implementar es:

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

- M: un arreglo de tamaño S describiendo el mensaje que Aisha quiere enviar a Basma.
- C: un arreglo de longitud 31 indicando los índices de los bits controlados por Cleopatra.
- Esta función puede ser llamada **a lo más 2100 veces** en cada caso de prueba.

Esta función debe llamar a la siguiente función para enviar un paquete:

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

- A: un paquete original (un arreglo de tamaño 31) representando los bits enviados por Aisha.
- Esta función retorna un paquete alterado B representando los bits que serán recibidos por Basma.
- Esta función puede ser llamada a lo más 100 veces en cada llamada de `send_message`.

La segunda función que debes implementar es:

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

- R: un arreglo que describe los paquetes alterados. Los paquetes provienen de los paquetes enviados por Aisha en una llamada a `send_message`, y aparecen **en el orden en el que fueron enviados** por Aisha. Cada elemento de R es un arreglo de longitud 31, representando un paquete alterado.
- Esta función debe retornar un arreglo de S bits que es igual al mensaje original M.
- Esta función puede ser llamada **varias veces** en cada caso de prueba, **exactamente una vez** para cada llamada correspondiente a `send_message`. El **orden de las llamadas a la función** `receive_message` no es necesariamente el mismo orden del correspondiente a las llamadas a `send_message`.

Nota que en el sistema del evaluador las funciones `send_message` y `receive_message` son llamadas en **dos programas separados**.

Restricciones

- $1 \leq S \leq 1024$

- C tiene exactamente 31 elementos, de los cuales 16 son iguales a 0 and 15 son iguales a 1.

Subtareas y Puntuaciones

Si en cualquiera de los casos de prueba, las llamadas a la función `send_packet` no cumplen con las reglas mencionadas arriba, o el valor retornado de cualquiera de las llamadas a la función `receive_message` es incorrecto, la puntuación para tu solución para ese caso de prueba será 0.

De lo contrario, sea Q el máximo número de llamadas a la función `send_packet` entre todas las llamadas a `send_message` sobre todos los casos de pruebas. Además, sea X igual a:

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

Luego, la puntuación es calculada como sigue:

Subtarea	Puntos	Restricciones Adicionales
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Sin restricciones adicionales.

Nota que en algunos casos el comportamiento del evaluador puede ser **adaptativo**. Es decir que los valores retornados por `send_packet` puede depender no solo de sus argumentos de entrada sino también de otras cosas, incluyendo las entradas y los valores de retorno a las llamadas a esta función y números pseudo-aleatorios generados por el evaluador. El evaluador es **determinístico** en el sentido que si lo ejecuta dos veces y en ambas ejecuciones envías los mismos paquetes, les realizará las mismas alteraciones a ellos.

Ejemplo

Considera la siguiente llamada:

```
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])
```

El mensaje que Aisha intenta enviar a Basma es $[0, 1, 1, 0]$. Los bits con indices desde 0 hasta 15 no puedes ser alterados por Cleopatra, mientras los bits con índices desde 16 hasta 30 pueden ser cambiados por Cleopatra.

Para este ejemplo, asumimos que Cleopatra llena los bits que controla de manera alternada con 0 y 1, es decir, le asigna 0 al primer índice que controla (índice 16 en este caso), 1 al segundo índice que controla (índice 17), 0 al tercer índice que ella controla (índice 18), y así sucesivamente.

Aisha puede decidir enviar dos bits del mensaje original en un paquete de la siguiente manera: envía el primer bit de 8 índices que controla y el segundo a los siguientes 8 índices que controla.

Luego, Aisha escoge enviar el siguiente paquete:

```
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])
```

Nota que Cleopatra puede cambiar los bits de los últimos 15 índices, entonces Aisha puede decidir qué poner en ellos arbitrariamente, pues Cleopatra puede sobreescribirlos. Con la estrategia de Cleopatra que asumimos en el ejemplo, la función devolverá $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Aisha decide enviar los últimos dos bits de M en el segundo paquete de una manera similar que en la anterior:

```
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])
```

Con la estrategia asumida a ser usada por Cleopatra, la función retorna: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha puede enviar más paquetes, pero ella escoge no enviar más.

El evaluador luego hace las siguientes llamadas a la función:

```
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, 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]])
```

Basma reconstruye el mensaje M de la siguiente forma. Para cada paquete, calcula el primer bit que aparece dos veces consecutivas y el último bit que aparece dos veces consecutivas. Entonces, en el primer paquete calcula los bits $[0,1]$ y en el segundo paquete calcula los bits $[1,0]$. Concatenando estos bits, reconstruye el mensaje $[0,1,1,0]$, que es el valor correcto que debe retornar para esta llamada a `receive_message`.

Se puede demostrar que con la estrategia de Cleopatra para este ejemplo, y para mensajes de longitud 4, este algoritmo utilizado por Basma reconstruye correctamente el mensaje M, sin importar el valor de C. Sin embargo, no es correcta en un caso general.

Evaluador de Ejemplo

El evaluador de ejemplo no es adaptativo: Cleopatra llena los bits consecutivos que controla con bits 0 y 1 alternadamente, como se describe en el ejemplo anterior.

Formato de entrada: **La primera línea contiene un entero T, que denota la cantidad de escenarios**. Luego aparecen T escenarios, cada uno de ellos en el siguiente formato:

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

Formato de salida: El evaluador de prueba muestra el resultado de cada uno de los T escenarios en el mismo orden en el que aparecen en la entrada, usando el siguiente formato:

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

Donde, K es el número de llamadas a `send_packet`, D es el mensaje retornado por `receive_message` y L es su tamaño.