Message

Aisha y Basma son dos amigos que suelen enviarse mensajes entre ellos. Aisha tiene un mensaje M que consiste de una secuencia de S bits (o sea, de unos y ceros), el cual quiere enviar a Basma. La forma en la que Aisha se comunica con Basma es enviándole **paquetes**. Un paquete es una secuencia de 31 bits, indexados del 0 al 30. Aisha le quiere enviar el mensaje M a Basma enviándole un cierto número de paquetes.

Lamentablemente, Cleopatra comprometió la comunicación entre Aisha y Basma y ahora es capaz de **alterar** los paquetes. Específicamente, en cada paquete Cleopatra puede modificar los bits en exactamente 15 índices. Formalmente hay un arreglo C de ceros y unos de longitud 31, que significa lo siguiente:

- Si $C[i] = 1$, entonces Cleopatra puede modificar el bit con índice i del paquete. Decimos que estos índices son **controlados** por Cleopatra.
- Si $C[i] = 0$, entonces Cleopatra no puede modificar el bit con índice i del paquete.

El arreglo C tiene exactamente 15 unos y 16 ceros. Al enviar el mensaje M, el conjunto de índices controlados por Cleopatra es el mismo para todos los paquetes. Aisha sabe exactamente cuáles 15 índices forman el conjunto controlado por Cleopatra. Basma solo sabe que 15 índices son controlados pero no sabe cuáles son.

Sea A un paquete que Aisha decide mandar (que denominaremos el **paquete original**). Sea B un paquete que recibe Basma (que denominaremos el **paquete alterado**). Para cada i con $0 \leq i < 31$:

- si Cleopatra no controla el bit con índice i (o sea, $C[i] = 0$), Basma recibe el bit i igual que lo mandó Aisha ($B[i] = A[i]$).
- de lo contrario, si Cleopatra sí controla el bit con índice i (es decir, $C[i] = 1$), entonces puede elegir el valor del bit $B[i]$.

Inmediatamente después de que Aisha envía un paquete, se entera de cual es el paquete alterado correspondiente.

Luego de que Aisha envía todos los paquetes, Basma recibe todos los paquetes alterados **en el orden en el que fueron enviados**, y debe reconstruir el mensaje original M.

Tu tarea es ingeniar e implementar una estrategia que permita a Aisha enviar 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 un arreglo C, y tiene que enviar algunos paquetes para comunicar 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: es el arreglo de longitud S que contiene el mensaje que Aisha quiere enviar a Basma.
- C: es el arreglo de longitud 31 que indica qué índices controla Cleopatra.
- Esta función se invocará **a lo sumo 2100 veces** en cada caso de prueba.

Para enviar un paquete, esa función tiene que llamar a la siguiente función

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

- A: el paquete original (un arreglo de longitud 31) que representa los bits enviados por Aisha.
- Esta función devuelve el paquete alterado B, que son los bits que recibe Basma.
- Esta función puede ser llamada **a lo sumo 100 veces** para cada invocación 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. Estos paquetes povienen de los paquetes enviados por Aisha con una llamada a `send_message`, y aparecen **en el orden en el que fueron enviados** por Aisha.
- Esta función debe devolver un arreglo de S bits que coincida con el mensaje original M.
- Esta función podrá ser invocada **múltiples veces** por cada caso de prueba, **exactamente una vez** para cada llamada correspondiente a `send_message`. El **orden de las llamadas a** `receive_message` no es necesariamente igual al orden de las llamadas correspondientes a `send_message`.

Nota que en el sistema de juzgado `send_message` y `receive_message` se ejecutan en **programas separados**.

Restricciones

- $1 \leq S \leq 1024$
- C tiene exactamente 31 elementos, de los cuales 16 son 0 y 15 son 1.

Subtareas y puntuación

Si en cualquiera de los casos de prueba, las llamadas a `send_packet` no cumplen con las reglas arriba mencionadas, o el valor que retorna cualquiera de las llamadas a `receive_message` es incorrecto, el puntaje para esa solución en ese caso de prueba será 0.

En caso contrario, sea Q el máximo número de llamadas a la función `send_packet` sobre todas las invocaciones a `send_message` en todos los casos de prueba, y sea X definido como

- $X = 1$, si $Q \leq 66$
- $X = 0.95^{Q-66}$, si $66 < Q \leq 100$
- $X = 0$, si $100 < Q$

Entonces el puntaje se calcula de la siguiente manera:

Subtarea	Puntaje	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 es **adaptativo**. Es decir, los valores que devuelve `send_packet` pueden depender de los valores de entrada y lo que hayan devuelto invocaciones anteriores a `send_packet`.

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 le intenta enviar a Basma es $[0, 1, 1, 0]$. Los bits con índices 0 a 15 no pueden ser modificados por Cleopatra, y los bits con índices 16 a 30 sí pueden ser modificados por Cleopatra.

Para el ejemplo, supongamos que el comportamiento de Cleopatra es determinístico, y que llena los bits que controla de manera alternada con ceros y unos. Es decir, le asigna 0 al primer índice

que controla (índice 16 en este caso), 1 al segundo índice que control (índice 17 en nuestro caso), 0 al tercer índice que controla (índice 18 en nuestro caso), y así sucesivamente.

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

Entonces Aisha mandará el siguiente mensaje

```
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 modificar los bits de los últimos 15 índices, entonces Aisha puede decidir qué poner en ellos arbitrariamente, pues Cleopatra puede sobreeescribirlos. Con la estrategia de Cleopatra que asumimos en el ejemplo, la función devolverá

$[0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Supongamos que Aisha decide enviar los siguientes dos bits de M de una manera similar:

```
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 que asumimos de Cleopatra, esta función devolverá $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha podría enviar más paquetes, pero en el ejemplo supongamos que no lo hace. Luego el evaluador hace la siguiente llamada

```
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 de la siguiente manera: para cada paquete, calcula el primer bit que aparece consecutivamente dos veces, y el último bit que aparece consecutivamente dos veces. Entonces, en el primer paquete calcula los bits $[0,1]$ y en el segundo paquete calcula los bits $[1,0]$. Concatenando estos bits, recupera el mensaje $[0,1,1,0]$, el cual es el valor correcto que debe retornar `receive_message`.

Se puede demostrar que con la estrategia de ejemplo de Cleopatra, y para mensajes de longitud 4, esta estrategia de Basma recupera correctamente el mensaje M, sin importar el valor de C. Sin embargo, no es correcta para el caso general.

Evaluador de Ejemplo

El evaluador de ejemplo no es adaptativo: el comportamiento de Cleopatra es determinístico, y 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 local 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 que devolvió `receive_message` y L es su longitud.