Mensaje

Aisha y Basma son dos amigas que se mandan mensajes la una a la otra. Aisha tiene el mensaje M, que es una secuencia de S bits que le quiere mandar a Basma. Aisha se comunica con Basma mandándole **paquetes**. Un paquete es una secuencia de 31 bits indexados de 0 a 30. Aisha quiere mandarle el mensaje M a Basma usando un número de paquetes.

Desafortunadamente, Cleopatra hackeó la comunicación entre Aisha y Basma y es capaz de **alterar** los paquetes. Esto significa que en cada paquete Cleopatra puede modificar bits en exactamente 15 índices. Específicamente, hay un arreglo C de longitud 31, en el que todos los elementos son 0 o 1, con el siguiente significado:

- Si $C[i] = 1$ quiere decir que el bit i puede ser modificado por Cleopatra. Llamaremos a esos índices **controlados** por Cleopatra.
- Si $C[i] = 0$ quiere decir que el bit i no puede ser modificado por Cleopatra.

El arreglo C contiene exactamente 15 unos y 16 ceros. Cuando se manda el mensaje M, los índices que son controlados por Cleopatra son los mismos para todos los paquetes. Aisha sabe exactamente cuáles 15 índices son controlados por Cleopatra. Basma solo sabe que 15 índices son controlados por Cleopatra, pero no cuáles.

Sea A un paquete que Aisha decide mandar (al cual llamaremos **paquete original**). Sea B el paquete recibido por Basma (al cual llamaremos **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 tal cual fue enviado por Aisha sin alteración ($B[i] = A[i]$),
- si no es así, y Cleopatra controla el bit con índice i ($C[i] = 1$), el valor de $B[i]$ lo decide Cleopatra.

Inmediatamente después de que Aisha manda cada paquete, ella sabe cuál es el **paquete alterado** que recibirá Basma.

Una vez que Aisha manda todos los paquetes, Basma recibe todos los paquetes alterados **en el mismo orden en el que fueron enviados**. y ella tiene que reconstruir el mensaje original M.

Tu tarea es idear e implementar una estrategia que permita a Aisha mandar el mensaje M a Basma, de tal forma que Basma pueda recuperar M basándose únicamente en los paquetes alterados. Específicamente, tienes que implementar dos funciones. La primera función realiza las

acciones de Aisha. Recibe el mensaje M y el arreglo C, y debe de enviar algunos paquetes para transmitir el mensaje a Basma. La segunda función realiza las acciones de Basma. Recibe los paquetes alterados y debe de recuperar el mensaje original M.

Detalles de implementación

La primera función que debes implementar es la siguiente:

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

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

Tu función debe de llamar a la siguiente función para mandar un paquete:

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

- A: el paquete original (un arreglo de longitud 31) representando los bits mandados por Aisha.
- Esta función regresa un paquete alterado B que representa los bits que serán enviados a Basma.
- Esta función la puedes llamar a lo más 100 veces por cada llamada a `send_message`.

La segunda función que debes implementar es:

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

- R: Un arreglo describiendo los paquetes alterados. Los paquetes se originan de los paquetes enviados por Aisha en una llamada a `send_message` y se reciben **en el mismo orden que fueron enviados** por Aisha. Cada elemento de R es un arreglo de longitud 31 representando un paquete alterado.
- Esta función debe de regresar un arreglo de S bits que debe de ser igual al mensaje original M.
- Esta función la va a llamar el evaluador **varias veces** en cada caso de prueba, pero **solamente una vez** por cada llamada a `send_message`. El **orden de las llamadas a** `receive_message` no es necesariamente el mismo orden que la llamada correspondiente a `send_message`.

Es importante notar que en el sistema de evaluación las llamadas a las funciones `send_message` y `receive_message` se hacen en **dos programas diferentes**.

Límites

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

Subtareas y evaluación

Si en cualquiera de los casos de prueba las llamadas a la función `send_packet` no siguen las reglas mencionadas arriba, o el valor que regresa las llamadas a tu función `receive_message` está mal, recibirás 0 puntos para ese caso de prueba.

En cualquier otro caso, sea Q el máximo número de llamadas a la función `send_packet` entre todas las llamadas a `send_message` a lo largo de todos los casos de prueba. También, sea X igual a:

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

Entonces, tus puntos se calculan de la siguiente forma:

Subtarea	Puntos	Condiciones adicionales
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Sin condiciones adicionales.

Es importante notar que en algunos casos, el comportamiento del evaluador puede ser **adaptativo**. Es decir, que los valores que son regresados por `send_packet` puede que dependan no solo de sus argumentos de entrada, sino también de muchas otras cosas, incluidas las entradas y los valores de retorno de las llamadas anteriores a este procedimiento y los números pseudoaleatorios generados por el calificador. El calificador es **determinista** en el sentido de que si lo ejecuta dos veces y en ambas ejecuciones envía los mismos paquetes, les hará los mismos cambios.

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 índices del 0 al 15 no pueden ser cambiados por Cleopatra, mientras que los índices del 16 al 30 sí pueden ser cambiados por Cleopatra.

Para los propósitos de este ejemplo, asumamos que Cleopatra llena los bits consecutivos que ella controla alternando 0 y 1, es decir, ella asigna 0 al primer bit que ella controla (índice 16 en nuestro caso), 1 al segundo bit que ella controla (índice 17), 0 al tercer bit que ella controla (índice 18), y así sucesivamente.

Aisha puede decidir enviar los primeros dos bits del mensaje original en un paquete de la siguiente forma: ella enviará el primer bit que ella controla de forma repetida en los primeros 8 bits que ella controla. y el segundo bit de manera repetida en los siguientes 8 bits que ella controla.

Aisha termina mandando 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])
```

Nótese que Cleopatra puede cambiar los bits en los últimos 15 índices, así Aisha los puede elegir arbitrariamente pues podrían ser sobrescritos. Con la estrategia de Cleopatra descrita, la función regresa: $[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]$.

Aisha decide mandar el segundo par de bits en M en un segundo paquete de 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 de Cleopatra, la función regresa: $[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1]$.

Aisha podría mandar más paquetes, pero decide no hacerlo.

El evaluador entonces llama a tu segunda función de la siguiente forma:

```
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 recupera el mensaje M de la siguiente forma. Para cada paquete que recibe, ella toma el primer bit que se repite dos veces seguidas, y el último bit que se repite dos veces seguidas. Es decir, del primer paquete, ella recupera los bits $[0, 1]$, y del segundo paquete recupera los bits $[1, 0]$.

Al juntarlos, ella es capaz de recuperar el mensaje original $[0, 1, 1, 0]$, que es el valor correcto para devolver en esta llamada a la función `receive_message`.

Se puede demostrar que con la estrategia descrita de Cleopatra y para mensajes de longitud 4, la solución de Basma recupera correctamente M, sin importar el valor de C. Sin embargo, esto no es correcto para un caso general.

Evaluador de ejemplo

El evaluador de ejemplo no es adaptativo, por el contrario Cleopatra llena bits consecutivos que ella controla con bits alternando 0 y 1, de la misma forma que en el caso de ejemplo.

Formato de entrada: **La primera línea de la entrada contiene un entero T, especificando el número de escenarios**. Después de eso, siguen T escenarios. Cada escenario tiene el siguiente formato

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

Formato de salida: El evaluador de ejemplo escribe el resultado de cada uno de los T escenarios en el mismo orden en el que están en la entrada y tiene el siguiente formato:

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

Aquí, K es el número de llamadas que se hizo a `send_packet`, D es el mensaje recibido por `receive_message` y L es su longitud.