Message

Aisha e Basma sono due amiche che si scambiano messaggi. Aisha ha un messaggio M, che è una sequenza di S bit (zero o uno), che vorrebbe inviare a Basma. Aisha comunica con Basma inviandole **pacchetti**, cioè sequenze di 31 bit indicizzate da 0 a 30. Aisha vorrebbe mandare il messaggio M a Basma inviandole un certo numero di pacchetti.

Sfortunatamente, Cleopatra ha compromesso la comunicazione tra Aisha e Basma ed è in grado di **corrompere** i pacchetti. Ciò significa che ci sono esattamente 15 indici che Cleopatra può modificare, se vuole, in ogni pacchetto. Nello specifico, esiste un array C di lunghezza 31, in cui ogni elemento è 0 o 1, con il seguente significato:

- $C[i] = 1$ indica che il bit con indice i può essere modificato da Cleopatra, e quindi lo chiamiamo **controllato** da Cleopatra.
- $C[i] = 0$ indica che il bit con indice i non può essere modificato da Cleopatra.

L'array C contiene esattamente 15 uni e 16 zeri. Durante l'invio del messaggio M, l'insieme degli indici controllati da Cleopatra rimane lo stesso per tutti i pacchetti. Aisha sa esattamente quali sono i 15 indici controllati da Cleopatra. Basma sa solo che Cleopatra controlla 15 indici, ma non sa quali.

Sia A un pacchetto che Aisha decide di inviare (detto **pacchetto originale**). Sia B il pacchetto ricevuto da Basma (detto **pacchetto corrotto**). Per ogni $0 \leq i < 31$:

- se Cleopatra non controlla il bit con indice i ($C[i] = 0$), Basma riceve il bit i inviato da Aisha ($B[i] = A[i]$),
- altrimenti, se Cleopatra controlla il bit con indice i ($C[i] = 1$), il valore di $B[i]$ è deciso da Cleopatra.

Subito dopo l'invio di ogni pacchetto, Aisha scopre qual è il pacchetto corrotto corrispondente.

Dopo che Aisha ha inviato tutti i pacchetti, Basma riceve tutti i pacchetti corrotti **nell'ordine in cui sono stati inviati** e deve ricostruire il messaggio originale M.

Il tuo compito è ideare e implementare una strategia che permetta ad Aisha di inviare il messaggio M a Basma, in modo che Basma possa recuperare M dai pacchetti corrotti. Nello specifico, devi implementare due funzioni. La prima funzione esegue le azioni di Aisha, che dato un messaggio M e l'array C, può inviare pacchetti per trasferire il messaggio a Basma. La seconda funzione esegue le azioni di Basma, che dati i pacchetti corrotti deve recuperare il messaggio originale M.

Note di implementazione

La prima funzione che devi implementare è:

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

- M: un array di lunghezza S che descrive il messaggio che Aisha vuole inviare a Basma.
- C: un array di lunghezza 31 che indica i bit controllati da Cleopatra.
- Questa funzione viene chiamata **al massimo 2100 volte** in ogni caso di test.

La funzione sopra può chiamare la seguente funzione per inviare un pacchetto:

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

- A: un pacchetto originale (un array di lunghezza 31) che rappresenta i bit inviati da Aisha.
- Questa funzione restituisce un pacchetto corrotto B che rappresenta i bit che saranno ricevuti da Basma.
- Questa funzione può essere chiamata al massimo 100 volte in ogni invocazione di `send_message`.

La seconda funzione da implementare è:

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

- R: un array che descrive i pacchetti corrotti. I pacchetti sono derivati da quelli inviati da Aisha nella chiamata a `send_message` corrispondente e sono forniti **nell'ordine in cui sono stati inviati** da Aisha. Ogni elemento di R è un array di lunghezza 31, che rappresenta un pacchetto corrotto.
- Questa funzione dovrebbe restituire un array di S bit che è uguale al messaggio originale M.
- Questa funzione viene chiamata **più volte** in ogni caso di test: **esattamente una volta** per ogni chiamata a `send_message` corrispondente. **L'ordine delle chiamate alla funzione** `receive_message` non è necessariamente lo stesso dell'ordine delle corrispondenti chiamate a `send_message`.

Nel sistema di valutazione le funzioni `send_message` e `receive_message` vengono chiamate in **due esecuzioni separate**.

Assunzioni

- $1 \leq S \leq 1024$
- C ha esattamente 31 elementi, di cui 16 sono uguali a 0 e 15 sono uguali a 1.

Subtask e punteggio

Se in uno qualsiasi dei casi di test, le chiamate alla funzione `send_packet` non sono conformi alle regole sopra menzionate, o il valore restituito da una qualsiasi delle chiamate alla funzione `receive_message` è errato, il punteggio della tua soluzione per quel caso di test sarà 0.

Altrimenti, sia Q il numero massimo di chiamate alla funzione `send_packet` tra tutte le invocazioni di `send_message` su tutti i casi di test. Sia inoltre X uguale a:

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

Quindi, il punteggio viene calcolato come segue:

Subtask	Punteggio	Limitazioni aggiuntive
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Nessuna limitazione aggiuntiva.

Si noti che in alcuni casi il comportamento del grader può essere **adattivo**. Ciò significa che i valori restituiti da `send_packet` possono dipendere non solo dai suoi argomenti di input ma anche da molti altri fattori, compresi gli input e i valori restituiti dalle chiamate precedenti a questa funzione e numeri pseudo-casuali generati dal grader. Il grader è **deterministico** nel senso che se lo esegui due volte e in entrambe le esecuzioni si inviano gli stessi pacchetti, verranno apportate le stesse modifiche.

Esempio

Consideriamo la seguente chiamata.

```
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])
```

Il messaggio che Aisha cerca di inviare a Basma è $[0, 1, 1, 0]$. I bit con indici da 0 a 15 non possono essere modificati da Cleopatra, mentre i bit con indici da 16 a 30 possono essere modificati da Cleopatra.

Supponiamo ad esempio che Cleopatra sovrascriva i bit che controlla alternando i bit 0 e 1, cioè assegnando 0 al primo indice che controlla (indice 16 nel nostro caso), 1 al secondo indice che controlla (indice 17), 0 al terzo indice che controlla (indice 18), e così via.

Aisha può decidere di inviare due bit del messaggio originale in un pacchetto come segue: assegnerà il primo bit ai primi 8 indici che controlla e il secondo bit ai seguenti 8 indici che controlla.

Aisha sceglie quindi di inviare il seguente pacchetto:

```
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 che Cleopatra può modificare gli ultimi 15 bit, quindi Aisha può impostarli come vuole, tanto potrebbero essere sovrascritti. Assumendo la strategia ipotizzata sopra per Cleopatra, la funzione restituisce: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Aisha decide di inviare gli ultimi due bit di M nel secondo pacchetto in modo simile a prima:

```
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 strategia ipotizzata sopra per Cleopatra, la funzione restituisce: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha potrebbe inviare altri pacchetti, ma sceglie di non farlo.

Il grader esegue quindi la seguente chiamata di funzione:

```
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 recupera il messaggio M come segue. Da ogni pacchetto prende il primo bit che si verifica due volte di seguito, e l'ultimo bit che si verifica due volte di seguito. Cioè, dal primo pacchetto prende i bit $[0,1]$, e dal secondo pacchetto prende i bit $[1,0]$. Mettendoli insieme, recupera il messaggio $[0,1,1,0]$ che è il valore corretto per questa chiamata a `receive_message`.

Si può dimostrare che con la strategia assunta da Cleopatra e per messaggi di lunghezza 4, questo approccio di Basma recupera correttamente M, indipendentemente dal valore di C. Tuttavia, nel caso generale questa strategia non è corretta.

Grader di esempio

Il grader di esempio non è adattivo. Inoltre, nel grader di esempio Cleopatra riempie sempre i bit consecutivi che controlla alternando 0 e 1, come descritto nell'esempio sopra.

Formato di input: **La prima riga dell'input contiene un intero T che specifica il numero di scenari.** Seguono T scenari. Ognuno di essi deve essere nel seguente formato:

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

Formato di output: Il grader scrive il risultato di ciascuno dei T scenari nello stesso ordine in cui sono forniti nell'input e nel seguente formato:

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

Qui, K è il numero di chiamate a `send_packet`, D è il messaggio restituito da `receive_message` e L è la sua lunghezza.