Nachricht

Die Freundinnen Aisha und Basma kommunizieren gerne über Nachrichten. Aisha hat eine Nachricht M, gegeben als Folge von S Bits (d.h. Nullen und Einsen), die sie an Basma verschicken möchte. Dazu kann Aisha mehrere **Pakete** an Basma verschicken. Ein **Paket** ist eine Folge von 31 Bits indiziert von 0 bis 30.

Cleopatra hat den Kommunikationskanal zwischen Aisha und Basma infiltriert und kann Pakete **verfälschen**. Cleopatra kann jedes Paket an 15 festgelegten Indizes abändern.

Sei C ein Array mit Länge 31, von dem jedes Element entweder 0 oder 1 ist.

- $C[i] = 1$ bedeutet, dass das i-te Bit von Cleopatra geändert werden kann. Wir sagen, dass diese Indizes von Cleopatra **kontrolliert** sind.
- $C[i] = 0$ bedeutet, dass das i-te Bit nicht von Cleopatra geändert werden kann.

Das Array C besteht aus genau 15 Einsen und 16 Nullen. Beim Verschicken von Nachricht M sind die Indizes, die von Cleopatra kontrolliert werden, gleich für alle Pakete. Aisha weiss genau, welche der 15 Indizes von Cleopatra kontrolliert werden. Basma weiss nur, dass 15 Indizes von Cleopatra kontrolliert werden, aber nicht welche.

Sei A das Paket, dass Aisha verschicken möchte (das **originale Paket**). Sei B das Paket, das von Basma empfangen wird (das **verfälschte Paket**). Für jedes i mit $0 \leq i < 31$:

- falls Cleopatra das i-te Bit nicht kontrolliert ($C[i] = 0$), dann empfängt Basma das i-te Bit wie von Aisha verschickt ($B[i] = A[i]$),
- ansonsten, falls Cleopatra das i-te Bit kontrolliert ($C[i] = 1$), kann Cleopatra den Wert von $B[i]$ frei wählen.

Direkt nachdem sie ein Paket verschickt hat, erfährt Aisha, wie das entsprechende verfälschte Paket aussieht.

Nachdem Aisha alle Pakete verschickt hat, empfängt Basma alle verfälschten Pakete **in der gleichen Reihenfolge, in der sie verschickt wurden** und muss damit die ursprüngliche Nachricht rekonstruieren.

Entwickle eine Strategie, die es Aisha erlaubt, Nachricht M an Basma so zu verschicken, dass Basma aus den verfälschten Paketen die ursprüngliche Nachricht M rekonstruieren kann. Konkret sollst du zwei Funktionen implementieren. Die erste Funktion führt die Aktionen von Aisha aus:

Gegeben Nachricht M und das Array C, verschicke Pakete um die Nachricht an Basma zu übertragen. Die zweite Funktion führt die Aktionen von Basma aus: Gegeben die verfälschten Pakete, rekonstruiere die ursprüngliche Nachricht M.

Angaben zur Implementierung

Die erste Funktion, die du implementieren sollst, ist:

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

- M: Ein Array der Länge S, das die Nachricht enthält, die Aisha an Basma verschicken möchte.
- C: Ein Array der Länge 31, das die Indizes der Bits angibt, die Cleopatra kontrolliert.
- Diese Funktion wird **bis zu 2100 Mal** pro Testfall aufgerufen.

Diese Funktion soll die folgende Funktion aufrufen um ein Paket zu verschicken:

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

- A: Ein originales Paket (ein Array der Länge 31) das die Bits, die von Aisha verschickt werden, repräsentiert.
- Diese Funktion gibt ein verfälschtes Paket B zurück, das die Bits, die von Basma empfangen werden, repräsentiert.
- Diese Funktion kann für jeden Aufruf von `send_message` bis zu 100 Mal aufgerufen werden.

Die zweite Funktion, die du implementieren sollst, ist:

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

- R: Ein Array, das die verfälschten Pakete beschreibt. Die Pakete wurden von Aisha in einem Aufruf von `send_message` geschickt und sind in **derselben Reihenfolge, in der sie von Aisha verschickt wurden**, in R enthalten. Jedes Element in R ist ein Array der Länge 31, das ein verfälschtes Paket repräsentiert.
- Diese Funktion soll ein Array mit S Bits zurückgeben, das mit der ursprünglichen Nachricht übereinstimmt.
- Diese Funktion kann **mehrfach** pro Testfall aufgerufen werden, **genau einmal** für jeden entsprechenden Aufruf von `send_message`. Die **Reihenfolge** der `receive_message`-**Funktionsaufrufe** ist nicht unbedingt die gleiche wie die Reihenfolge der `send_message` Aufrufe.

Beachte, dass im Grading-System die Funktionen `send_message` und `receive_message` in zwei **verschiedenen Programmen** aufgerufen werden.

Einschränkungen

- $1 \leq S \leq 1024$
- C hat genau 31 Elemente, von denen 16 gleich 0 und 15 gleich 1 sind.

Subtasks und Punkte

Ein Testfall wird mit 0 Punkten bewertet, wenn das Aufrufen der Funktion `send_packet` nicht den obigen Regeln entspricht, oder der Rückgabewert von einem der Funktionsaufrufe von `receive_message` falsch ist.

Sei Q die maximale Anzahl an Aufrufen der Funktion `send_packet` über alle Funktionsaufrufe von `send_message` und alle Testfälle. Dann ist X gegeben durch:

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

Die Punkte werden wie folgt berechnet:

Subtask	Punkte	Weitere Einschränkungen
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Keine weiteren Einschränkungen.

Anmerkung: In einigen Fällen kann sich der Grader **adaptiv** verhalten. Also können die Rückgabewerte von `send_packet` nicht nur von den Eingabewerten abhängen, sondern auch von anderen Werten, wie den Eingabe- und Rückgabewerten von vorherigen Aufrufen der Funktion oder vom Grader generierte Pseudo-Zufallszahlen. Der Grader ist **deterministisch** im Sinne, dass wenn zwei verschiedene Durchläufe mit den gleichen Parametern laufen und du die gleichen Pakete schickst, er dann die gleichen Verfälschungen vornehmen wird.

Beispiel

Betrachte den folgenden Funktionsaufruf.

```
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])
```

Die Nachricht, die Aisha versucht, an Basma zu verschicken ist $[0, 1, 1, 0]$. Die Bits mit Indizes von 0 bis 15 können nicht von Cleopatra verfälscht werden, aber die Bits mit Indizes von 16 bis 30 können verfälscht werden.

Für dieses Beispiel nehmen wir an, dass Cleopatras aufeinanderfolgende Bits, die sie verfälschen kann, abwechselned mit 0-en und 1-en füllt. Das heißt, sie setzt den ersten Bit, den sie verfälschen kann, auf 0 (in unserem Fall Index 16), sie setzt den zweiten auf 1 (Index 17), den dritten auf 0 (Index 18), und so weiter.

Aisha kann sich dazu entscheiden, zwei Bits der ursprüglichen Nachricht folgendermaßen in einem Paket zu verschicken: Sie verschickt den ersten Bit in den ersten 8 Indizes, die sie kontrolliert, und den zweiten Bit in den nächsten 8 Indizes.

Dann schickt Aisha folgendes Paket

```
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])
```

Beachte, dass Cleopatra nur die Bits der letzten 15 Indizes verfälschen kann. Also kann Aisha diese Bits willkürlich setzen, da sie überschrieben werden können. Mit der angenommenen Strategie von Cleopatra wäre der Rückgabewert der Funktion: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha entscheidet sich nun dazu, die letzten zwei Bits von M ähnlich wie zuvor zu verschicken:

```
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])
```

Mit der angenommenen Strategie von Cleopatra, wäre der Rückgabewert der Funktion: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha könnte noch weitere Pakete verschicken, aber sie entscheidet sich, dies nicht zu machen.

Der Grader macht nun folgenden Funktionsaufruf:

```
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 rekonstruiert die Nachricht M wie folgt. Von jedem Paket nimmt sie das erste Bit, das zweimal hintereinander vorkommt und das letzte Bit, das zweimal hintereinander vorkommt. Also erhält sie aus dem ersten Paket die Bits $[0,1]$ und aus dem zweiten Paket die Bits $[1,0]$. Durch Zusammenfügen erhält sie die Nachricht $[0,1,1,0]$, welches der richtige Rückgabewert dieses Funktionsaufrufes von `receive_message` ist.

Man kann für diese angenommene Strategie von Cleopatra und Nachrichten der Länge 4 zeigen, dass diese Methode von Basma alle Nachrichten M korrekt rekonstruiert, unabhängig von den Werten von C.

Dies ist aber im Allgemeinen nicht richtig.

Beispielgrader

Der Beispielgrader ist nicht adaptiv. Das Verhalten von Cleopatra ist also deterministisch. Sie füllt aufeinanderfolgende Bits, die sie verfälschen kann, abwechselnd mit 0-en und 1-en, wie im Beispiel oben beschrieben.

Eingabeformat: **Die erste Zeile der Eingabe enhält eine ganze Zahl T, die die Anzahl der Szenarien angibt.** Es folgen T Szenarien. Jedes wird in folgendem Format angegeben:

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

Ausgabeformat: Der Beispielgrader gibt das Ergebnis von jedem der T Szenarien in der gleichen Reihenfolge, wie bei der Eingabe, in folgendem Format aus:

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

Dabei ist K die Anzahl der Funktionsaufrufe von `send_packet`, D die Nachricht, die von `receive_message` zurückgegeben wurde, und L die Länge dieser Nachricht.