Message

Aisha et Basma sont deux amis qui communiquent ensemble. Aisha a un message M, qui est une séquence de S bits (de zéros et de uns), qu'elle voudrait envoyer à Basma. Aisha communique avec Basma en lui envoyant des **paquets**. Un paquet est une séquence de 31 bits indexés de 0 à 30. Aisha souhaiterait envoyer le message M à Basma en lui envoyant un certain nombre de paquets.

Malheureusement, Cléopâtre a compromis la communication entre Aisha et Basma, et est capable de **corrompre** les paquets. Cléopâtre peut modifier la valeur des bits à 15 indices du paquet. Plus précisément, il y a un tableau C de longueur 31, dans lequel chaque élément est soit 0 soit 1, ayant la signification suivante :

- $C[i] = 1$ indique que le bit d'indice i peut être modifié par Cléopâtre. On dit que ces indices sont **contrôlés** par Cléopâtre.
- $C[i] = 0$ indique que le bit d'indice i ne peut pas être modifié par Cléopâtre.

Le tableau C contient exactement 15 uns et 16 zéros. Pendant l'envoi du message M, l'ensemble des indices contrôlés par Cléopâtre reste le même pour tous les paquets. Aisha connaît exactement quels sont les 15 indices contrôlés par Cléopâtre. Basma sait seulement que 15 bits sont contrôlés par Cléopâtre, mais elle ne connaît pas leurs indices.

Soit A un paquet que Aisha envoie (que l'on appellera le **paquet original**). Soit B le paquet qui est reçu par Basma (que l'on appellera le **paquet corrompu**).

Pour tout i tel que $0 \leq i < 31$:

- si Cléopâtre ne contrôle pas le bit d'indice i ($C[i] = 0$), alors Basma reçoit le bit i envoyé par Aisha ($B[i] = A[i]$),
- si Cléopâtre contrôle le bit d'indice i ($C[i] = 1$), alors la valeur de $B[i]$ est choisie par Cléopâtre.

Immédiatement après avoir envoyé chaque paquet, Aisha observe le paquet corrompu correspondant.

Après qu'Aisha a envoyé tous les paquets, Basma reçoit tous les paquets corrompus **dans le même ordre qu'à l'envoi** et doit reconstruire le message M.

Votre tâche est de concevoir et implémenter une stratégie qui permettrait à Aisha d'envoyer le message M à Basma, tel que Basma puisse reconstruire M à partir des paquets corrompus. Plus précisément, vous devez implémenter deux fonctions. La première fonction effectue les actions

d'Aisha. Elle reçoit en paramètre le message M et le tableau C, et doit renvoyer les paquets à transférer à Basma. La deuxième fonction effectue les actions de Basma. Elle reçoit en paramètre les paquets corrompus, et doit reconstruire le message M.

Détails d'implémentation

La première fonction que vous devez implémenter est :

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

- M : un tableau de longueur S décrivant le message qu'Aisha souhaite envoyer à Basma.
- C : un tableau de longueur 31 indiquant les indices des bits contrôlés par Cléopâtre.
- Cette procédure sera appelée **au plus 2100 fois** dans chaque test.

Cette fonction doit appeler la fonction suivante pour envoyer un paquet :

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

- A : un paquet original (un tableau de longueur 31) représentant les bits envoyés par Aisha.
- Cette fonction renvoie un paquet corrompu B représentant les bits reçus par Basma.
- Cette fonction peut être appelée au plus 100 fois dans chaque appel à `send_message`.

La deuxième fonction que vous devez implémenter est :

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

- R : un tableau décrivant les paquets corrompus. Les paquets proviennent de paquets envoyés par Aisha dans un appel à la fonction `send_message` et sont fournis **dans l'ordre d'envoi** par Aisha. Chaque élément de R est un tableau de longueur 31, représentant un paquet corrompu.
- Cette fonction doit renvoyer un tableau de S bits qui est égal au message M.
- Cette fonction sera appelée **plusieurs fois** dans chaque test, et **exactement une fois** pour chaque appel à `send_message`. L'**ordre des appels à la fonction** `receive_message` n'est pas forcément le même que l'ordre des appels à `send_message` correspondants.

Notez que dans le système d'évaluation, les fonctions `send_message` et `receive_message` sont appelés dans **deux programmes séparés**.

Contraintes

- $1 \leq S \leq 1024$
- C a exactement 31 éléments, parmi lesquels 16 sont égaux à 0 et 15 sont égaux à 1.

Sous-tâches et Score

Si dans un des tests, les appels à la fonction `send_packet` ne respectent pas les règles mentionnées ci-dessus, ou si la valeur renvoyée par un des appels à la fonction `receive_message` est incorrecte, alors le score de votre de votre solution pour ce test sera de 0.

Sinon, soit Q le nombre maximal d'appels à la fonction `send_packet` parmi tous les appels à `send_message` dans tous les tests. Aussi, soit X égal à :

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

Alors, le score est calculé de la manière suivante :

Sous-tâche	Score	Contrainte supplémentaire
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Aucune contrainte supplémentaire.

Notez que dans certaines situations, le comportement du grader peut être **adaptatif**. Cela signifie que les valeurs renvoyées par `send_packet` peuvent dépendre non-seulement des arguments d'entrée mais aussi d'autres choses comme les arguments d'entrée et valeurs de retour des appels précédents à cette fonction, ou des nombres pseudo-aléatoires générés par le grader. Le grader est **déterministe** dans le sens où si on effectue deux exécutions en envoyant les même paquets, il les modifiera de la même manière.

Exemple

Considérons l'appel suivant.

```
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])
```

Le message qu'Aisha essaie d'envoyer est $[0, 1, 1, 0]$. Les bits d'indices 0 à 15 ne peuvent pas être modifiés par Cléopâtre, tandis que les bits d'indices 16 à 30 peuvent être modifiés par Cléopâtre.

Dans cet exemple, supposons que Cléopâtre remplit les bits consécutifs qu'elle contrôle en alternant entre 0 et 1, i.e. elle affecte 0 au premier indice qu'elle contrôle (indice 16 dans notre cas), 1 au second indice qu'elle contrôle (indice 17), 0 au troisième indice qu'elle contrôle (indice 18), etc.

Aisha peut décider d'envoyer deux bits du message dans un paquet comme suit : elle enverra le premier bit aux 8 premiers indices qu'elle contrôle et le second bit aux 8 indices suivants qu'elle

contrôle.

Aisha choisira alors d'envoyer le paquet suivant :

```
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])
```

Notez que Cléopâtre peut modifier les bits des 15 derniers indices, donc Aisha peut les fixer arbitrairement, car il seront modifiés. Avec l'hypothèse faite sur la stratégie de Cléopâtre, la fonction renvoie : $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Aisha décide d'envoyer les deux derniers bits de M dans le second paquet de la même manière que précédemment :

```
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])
```

Avec l'hypothèse faite sur la stratégie de Cléopâtre, la fonction renvoie : $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha pourrait envoyer plus de paquets, mais décide de ne pas le faire.

Le grader fait l'appel de fonction suivant :

```
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 reconstruit le message M de la manière suivante. De chaque paquet reçu elle prend le premier bit qui apparaît deux fois d'affilée, et le dernier bit qui apparaît deux fois d'affilée. C'est à dire, elle prend les bits $[0,1]$ du premier paquet, et elle prend les bits $[1,0]$ du second paquet. En les rassemblant, elle reconstruit le message $[0,1,1,0]$, qui correspond à la valeur de retour correcte pour cet appel à `receive_message`.

Il peut être démontré qu'avec l'hypothèse faite sur la stratégie de Cléopâtre et pour tous messages de longueur 4, cette approche de Basma reconstruit correctement M, quelle que soit la valeur de C. Cependant, cette approche n'est pas correcte dans le cas général.

Évaluateur d'exemple (Grader)

L'évaluateur d'exemple n'est pas adaptatif. À la place, Cléopâtre remplit les bits consécutifs qu'elle contrôle en alternant entre 0 et 1, comme décrit dans l'exemple ci-dessus.

Format d'entrée : **La première ligne de l'entrée contient un entier T, spécifiant le nombre de scénarios.** T scénarios suivent. Chacun d'entre eux est fourni dans le format suivant :

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

Format de sortie : L'évaluateur d'exemple écrit le résultat de chacun des T scénarios, dans le même ordre que celui de l'entrée, et au format suivant :

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

Ici, K est le nombre d'appels à `send_packet`, D est le message renvoyé par `receive_message` et L est sa longueur.