Повідомлення

Айша і Басма — дві подруги, які листуються між собою. Айша має повідомлення M, яке є послідовністю S бітів (тобто нулів або одиниць), що вона хотіла б надіслати Басмі. Айша спілкується з Басмою, надсилаючи їй **пакети**. Пакет — це послідовність з 31 бітів з індексами від 0 до 30. Айша хоче надіслати Басмі повідомлення M, надіславши їй певну кількість пакетів.

На жаль, Клеопатра скомпрометувала спілкування між Айшею та Басмою і здатна **зіпсувати** пакети. Тобто в кожному пакеті Клеопатра може модифікувати біти в рівно 15 індексах. Зокрема, існує масив C довжиною 31, у якому кожен елемент є або 0 або 1, що має таке значення:

- $C[i] = 1$ вказує, що біт з індексом i може бути змінений Клеопатрою. Ми називаємо ці індекси **контрольованими** Клеопатрою.
- $C[i] = 0$ вказує, що біт з індексом i не може бути змінений Клеопатрою.

Масив C містить рівно 15 одиниць і 16 нулів. Під час надсилання повідомлення M, набір індексів, керованих Клеопатрою, залишається незмінним для всіх пакетів. Айша точно знає, які 15 індексів контролює Клеопатра. Басма знає лише те, що 15 індексів контролює Клеопатра, але вона не знає, які індекси.

Нехай A — пакет, який вирішила надіслати Айша (який ми називаємо **оригінальним пакетом**). Нехай B — пакет, отриманий Басмою (який ми називаємо **зіпсованим пакетом**). Для кожного i такого, що $0 \leq i < 31$:

- якщо Клеопатра не контролює біт з індексом i ($C[i] = 0$), Басма отримує біт i надісланий Айшею ($B[i] = A[i]$),
- інакше, якщо Клеопатра контролює біт з індексом i ($C[i] = 1$), значення $B[i]$ визначається Клеопатрою.

Одразу після надсилання кожного пакета, Айша дізнається відповідний зіпсований пакет.

Після того як Айша надішле всі пакети, Басма отримує всі зіпсовані пакети **в порядку їх надсилання** і має відновити оригінальне повідомлення M.

Ваше завдання — розробити та реалізувати стратегію, що дозволить Айші надіслати повідомлення M Басмі, щоб Басма могла відновити M із зіпсованих пакетів. Зокрема, вам слід реалізувати дві функції. Перша функція виконує дії Айші. Дано повідомлення M і масив

C, і функція має надіслати кілька пакетів для передачі повідомлення Басмі. Друга функція виконує дії Басми. Отримує зіпсовані пакети і має відновити оригінальне повідомлення M .

Деталі реалізації

Перша функція, яку ви повинні реалізувати:

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

- M: описує масив довжини S повідомлення, яке Айша хоче надіслати Басмі.
- C: масив довжиною 31 із зазначенням індексів бітів, якими керує Клеопатра.
- Цю функцію можуть викликати **не більше 2100 разів** у кожному тестовому випадку.

Ця функцію повинна викликати наступну функцію для надсилання пакета:

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

- A: оригінальний пакет (масив довжиною 31) що представляє біти, надіслані Айшею.
- Ця функція повертає зіпсований пакет B, що представляє біти, які буде отримано Басмою.
- Цю функцію можна викликати не більше 100 разів у кожному виклику `send_message`.

Друга функція, яку ви повинні реалізувати:

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

- R: масив, що описує зіпсовані пакети. Пакети походять із пакетів, надісланих Айшею під час певного виклику `send_message`, і надаються в тому порядку, у якому вони були надіслані Айшею. Кожен елемент R — це масив довжиною 31, що представляє зіпсований пакет.
- Ця функція має повернути масив з S бітів, який дорівнює вихідному повідомленню M.
- Цю функцію можуть викликати **кілька разів** у кожному тестовому випадку, **рівно один раз** для кожного відповідного виклику `send_message`. Порядок викликів функції `receive_message` не обов'язково збігається з порядком відповідних викликів `send_message`.

Зверніть увагу, що в системі тестування функції `send_message` і `receive_message` викликаються в двох окремих програмах.

Обмеження

- $1 \leq S \leq 1024$

- C містить рівно 31 елемент, з яких 16 дорівнюють 0, а 15 дорівнюють 1.

Підзадачі та оцінювання

Якщо в будь-якому з тестових випадків, виклики функції `send_packet` не відповідають правилам, згаданим вище, або значення, що повертається будь-яким із викликів функції `receive_message` є неправильним, оцінка вашого рішення для цього тесту буде 0.

В іншому випадку, нехай Q - це максимальна кількість викликів функції `send_packet` серед усіх викликів `send_message` у всіх тестових випадках. Також нехай X дорівнює:

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

Тоді оцінка розраховується наступним чином:

Підзадача	Балів	Додаткові обмеження
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Без додаткових обмежень

Зверніть увагу, що в деяких випадках поведінка градера може бути **адаптивною**. Це означає, що значення, які повертає `send_packet` може залежати не лише від її вхідних аргументів, а й від багатьох інших речей, включаючи вхідні дані та значення, що повертаються попередніми викликами цієї функції та псевдовипадковими числами, згенеровані градером. Градер є **детермінованим** у тому сенсі, що якщо ви запустите його двічі і під час обох запусків ви надсилаєте однакові пакети, він внесе в них однакові зміни.

Приклади

Розглянемо наступний виклик.

```
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])
```

Повідомлення, яке Айша намагається надіслати Басмі, це $[0, 1, 1, 0]$. Біти з індексами від 0 до 15 не можуть бути змінені Клеопатрою, а біти з індексами від 16 до 30 можуть бути змінені Клеопатрою.

Для цього прикладу припустимо, що Клеопатра замінює послідовні біти, що вона контролює, чергуючи 0 і 1, тобто вона надає значення 0 до першого індексу, який вона контролює (індекс

16 у нашому випадку), 1 до другого індексу, який вона контролює (індекс 17), 0 до третього індексу, який вона контролює (індекс 18), і так далі.

Айша може вирішити надіслати два біти з оригінального повідомлення в одному пакеті наступним чином: вона надішле перший біт за першими 8 індексами, якими вона керує а другий біт у наступних 8 індексах, якими вона керує.

Потім Айша вирішує надіслати такий пакет:

```
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])
```

Зверніть увагу, що Клеопатра може змінювати біти з останніми 15 індексами, тому Айша може встановити їх довільно, оскільки вони можуть бути перезаписані. З передбачуваною стратегією Клеопатри функція повертає: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Айша вирішує надіслати останні два біти M у другому пакеті таким же чином, як і раніше:

```
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])
```

З передбачуваною стратегією Клеопатри функція повертає: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Айша може надсилати більше пакетів, але вона вирішує цього не робити.

Потім градер виконує наступний виклик функції:

```
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]])
```

Басма відновлює повідомлення M наступним чином. З кожного пакета вона бере перший біт, який зустрічається двічі поспіль, і останній біт, який зустрічається двічі поспіль. Тобто з першого пакету вона бере біти $[0,1]$, а з другого пакету вона приймає біти $[1,0]$. Зібравши їх разом, вона відновлює повідомлення $[0,1,1,0]$, що є правильним значенням, що повертається для цього виклику `receive_message`.

Можна показати, що з цією стратегією Клеопатри та для повідомлень довжиною 4, цей підхід Басми правильно відновлює M, незалежно від значення C. Однак у загальному випадку це не правильно.

Приклад градера

Приклад градера не є адаптивним. Натомість Клеопатра заповнює послідовні біти, які вона контролює, чергуючи біти 0 і 1, як описано в прикладі вище.

Формат вхідних даних: **Перший рядок містить ціле число T – кількість сценаріїв.** Далі слідує опис цих T сценаріїв. Кожен з них надається в такому форматі:

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

Формат вихідних даних: Приклад градера записує результат кожного з T сценаріїв у такому самому порядку, як вони надані у вхідних даних у такому форматі:

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

Тут K – це кількість викликів до `send_packet`, D – це повідомлення, яке повертає `receive_message` і L – це його довжина.