Message

Айша и Басма — друзья, которые переписываются друг с другом. У Айши есть сообщение M, которое является последовательностью из S битов (нулей и единиц), которое она хочет передать Басме. Для этого Айша отправляет Басме **пакеты**. Пакет — это последовательность из 31 бита, пронумерованных от 0 до 30. Айша хочет передать сообщение M Басме, отправив ей некоторое количество пакетов.

К сожалению, Клеопатра вмешивается в канал связи между Айшей и Басмой и может **портить** пакеты. В каждом пакете Клеопатра имеет возможность изменять биты на ровно 15 индексах. Определим массив C длины 31, такой что в нем каждый элемент это 0 или 1, в котором:

- $C[i] = 1$ означает, что бит с индексом i может быть изменен Клеопатрой. Назовем такие индексы **контролируемыми** Клеопатрой.
- $C[i] = 0$ означает, что бит с индексом i не может быть изменен Клеопатрой.

Массив C содержит ровно 15 единиц и 16 нулей. При передаче одного сообщения, множество индексов, контролируемых Клеопатрой, остается одинаковым для всех пакетов. Айша знает, какие 15 индексов контролируются Клеопатрой. Басма знает только что ровно 15 индексов контролируются Клеопатрой, но она не знает эти индексы.

Пусть A это пакет, который Айша решает отправить (назовем его **изначальным пакетом**). Пусть B это пакет, полученный Басмой (назовем его **испорченным пакетом**). Для каждого i, такого что $0 \leq i < 31$:

- если Клеопатра не контролирует индекс i ($C[i] = 0$), тогда Басма получает бит i, который послала Айша ($B[i] = A[i]$),
- иначе, если Клеопатра контролирует бит с индексом i ($C[i] = 1$), значение $B[i]$ определяет Клеопатра.

Немедленно после отправки каждого пакета Айша узнает, какой испорченный пакет получился из него.

После того, как Айша отправит все пакеты, Басма получит испорченные пакеты **в порядке, в котором они были отправлены** и должна восстановить сообщение M.

Ваша задача — разработать и реализовать стратегию, которая позволит Айше передать сообщение M Басме так, чтобы Басма смогла восстановаить M из испорченных пакетов.

Для этого вам нужно реализовать две функции. Первая функция выполняет действия Айши. Ей дано сообщение M и массив C, она должна отправить некоторые пакеты, чтобы передать сообщение Басме. Вторая функция выполняет действия Басмы. Ей даются испорченные пакеты и она должна восстановить изначальное сообщение M.

Implementation Details

Первая функция, которую вы должны реализовать:

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

- M: массив длины S, определяющий сообщение, которое Айша хочет передать Басме.
- C: массив длины 31, определяющий индексы, контролируемые Клеопатрой.
- Эта функция может быть вызвана **не более 2100 раз** для каждого теста.

Внутри этой функции вы должны использовать следующую функцию, чтобы отправить пакет:

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

- A: это изначальный пакет (массив длины 31), состоящий из битов, отправленных Айшей.
- Эта функция возвращает массив B, состоящий из битов, которые получит Басма.
- Эта функция может быть вызвана не более 100 раз для каждого вызова `send_message`.

Вторая функция, которую вы должны реализовать:

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

- R: массив, описывающий полученные испорченные пакеты. Пакеты соответствуют пакетам, оправленным Айшей в одном из вызовов `send_message` и даны **в порядке, в котором они были отправлены** Айшей. Каждый элемент R это массив длины 31, описывающий испорченный пакет.
- Эта функция должна вернуть массив из S битов, который равен изначальному сообщению M.
- Эта функция может быть вызвана **несколько раз** для каждого теста, **ровно один раз** для каждого соответствующего вызова `send_message`. **Порядок вызовов функции** `receive_message` не обязательно совпадает с порядком соответствующих вызовов функции `send_message`.

Обратите внимание, что в тестирующей системе функции `send_message` и `receive_message` вызываются в **различных запусках программы**.

Constraints

- $1 \leq S \leq 1024$
- C содержит ровно 31 элемент, из которых 16 это 0 и 15 это 1.

Subtasks and Scoring

Если в каком-то из тестов хотя бы один вызов функции `send_packet` не удовлетворяет правилам, описанным выше, или хотя бы одно возвращенное значение функции `receive_message` неверное, балл решения за этот тест будет 0.

Иначе, пусть Q это максимальное количество вызовов функции `send_packet` среди всех вызовов `send_message` во всех тестах. Также пусть X равно:

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

Тогда балл вычисляется следующим образом:

Подзадача	Балл	Дополнительные ограничения
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Нет дополнительных ограничений

Обратите внимание, что в некоторых случаях поведение грейдера может быть **адаптивным**. Это означает, что значения, возвращаемые `send_packet`, могут зависеть не только от входных значений, но и от многих других вещей, включая входные данные и возвращаемые значения предыдущих вызовов этой функции и псевдослучайные числа, сгенерированные грейдером. Грейдер **детерминирован** в том смысле, что если вы запустите его дважды, и в обоих запусках вы отправите одни и те же пакеты, он внесёт в них одни и те же изменения.

Example

Рассмотрим следующий вызов.

```
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])
```

Сообщение, которое Айша хочет передать Басме это $[0, 1, 1, 0]$. Биты с индексами от 0 до 15 не могут быть изменены Клеопатрой, а биты с индексами от 16 до 30 могут быть изменены Клеопатрой.

Для простоты описания этого примера, допустим что Клеопатра заполняет биты, которые она контролирует, чередуя значения 0 и 1. Таким образом, она выставляет бит 0 первому индексу, который она контролирует (индекс 16 в этом примере), 1 второму индексу, который она контролирует (индекс 17), 0 третьему индексу, который она контролирует (индекс 18) и так далее.

Айша может решить отправить два бита из изначального сообщения в одном пакете следующим образом: она отправит первый бит в первых 8 индексах, которые она контролирует и второй бит в следующих 8 индексах, которые она контролирует.

Айша отправляет следующий 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])
```

Обратите внимание, что Клеопатра может менять биты на последних 15 индексах, таким образом Айша может ставить их значения любыми, потому что они могут быть перезаписаны. При описанном поведении Клеопатры функция вернет: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Айша решает отправить два последних бита M во втором пакете таким же образом, как два бита до этого:

```
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])
```

При описанном поведении Клеопатры функция вернет: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Айша может продолжать отправлять пакеты, но решает остановиться.

Затем грейдер делает следующий вызов функции:

```
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 восстанавливает сообщение M следующим образом. Из каждого пакета она берет первый бит, который встречается дважды подряд в массиве и последний бит, который встречается дважды подряд в массиве. Таким образом, из первого пакета она берет биты $[0,1]$ и из второго пакета она берет биты $[1,0]$. Соединяя их она получает сообщение

$[0, 1, 1, 0]$, которое является правильным возвращаемым значением на этот вызов функции `receive_message`.

Можно показать, что в предположении того, что Клеопатра действует в соответствии с описанным поведением и сообщения имеют длину 4, эта стратегия позволяет Басме верно восстановить M, независимо от значения C. Хотя эта стратегия не является верной в общем случае.

Sample Grader

Пример грейдера не является адаптивным. В нем Клеопатра заполняет контролируемые биты, чередуя значения 0 и 1, как описано в примере выше.

Input format: В первой строке находится целое число T — количество запусков в тесте. Далее следуют описания T запусков, каждое из них дано в следующем формате:

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

Output format: Пример грейдера выводит результат на каждом из T запусков в том же порядке, в котором они даны во входных данных в следующем формате:

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

Здесь K это количество вызовов функции `send_packet`, D это сообщение, которое вернула функция `receive_message` и L это его длина.