# Съобщение

Айша и Басма са две приятелки, които комуникират една с друга. Айша има съобщение $M$, което е редица от $S$ бита (т.е. нули или единици). Тя иска да го изпрати на Басма. Айша комуникира с Басма, като ѝ изпраща **пакети**. Пакет е редица от $31$ бита, индексирани с числата от $0$ до $30$. Айша иска да изпрати съобщението $M$ на Басма, като ѝ изпраща някакъв брой пакети.

За съжаление, Клеопатра иска да компрометира комуникацията между Айша и Басма, като **потенциално променя** пакетите. Във всеки пакет, Клеопатра може да промени битове на точно $15$ индекса. По-конкретно, има редица $C$ с дължина $31$, на който всеки елемент е или $0$, или $1$, със следното значение:

- $C[i] = 1$ означава, че битът с индекс $i$ може да бъде променен от Клеопатра. Казваме, че тези индекси са **контролирани** от Клеопатра.
- $C[i] = 0$ означава, че битът с индекс $i$ не може да бъде променен от Клеопатра.

Редицата $C$ съдържа точно $15$ единици и $16$ нули. Докато се изпращат пакети за съобщението $M$, множеството от индекси, контролирани от Клеопатра, остава същото за всички пакети. Айша знае точно кои $15$ индекса са контролирани от Клеопатра. Басма единствено знае, че има някои $15$ индекса, които са контролирани от Клеопатра, но не знае точно кои индекси.

Нека $A$ е пакет, изпратен от Айша (който ще наричаме **оригинален пакет**). Нека $B$ е пакет, получен от Басма (който ще наричаме **потенциално променен пакет**, възможно е да съвпада с оригиналния пакет). За всяко $i$, така че $0 \leq i < 31$:

- ако Клеопатра не контролира бита с индекс $i$ ($C[i] = 0$), то Басма ще получи бита $i$ точно както е изпратен от Aisha ($B[i] = A[i]$),
- иначе, ако Клеопатра контролира бита с индекс $i$ ($C[i] = 1$), то стойността на $B[i]$ ще бъде определена от Клеопатра.

Веднага след като изпрати всеки пакет, Айша научава точно какъв е съответният изпратен потенциално променен пакет.

След като Айша изпрата всички пакети, Басма получава всички потенциално променени пакети **в реда, в който са изпратени** и трябва да възстанови съобщението $M$.

Вашата задача е да съставите и имплементирате стратегия, която ще позволи на Айша да изпрати съобщение $M$ на Басма, така че Басма да може да възстанови съобщението $M$ от потенциално променените пакети. По-конкретно, трябва да напишете две функции. Първата функция изпълнява дейстията на Айша. Това означава, че се подават съобщение $M$ и редица $C$, и трябва да се изпратят някакви пакети, за да се предаде съобщението на Басма. Втората функция трябва да изпълнява действията на Басма. Това означава, че се подават потенциално променените пакети и трябва да се възстанови оригиналното съобщение $M$.

## Детайли по имплементацията

Първата функция, която трябва да имплементирате, е следната:

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

- $M$: вектор с дължина $S$, който представлява съобщението, което Айша иска да изпрати на Басма.
- $C$: вектор с дължина $31$, който описва индексите на битовете, контролирани от Клеопатра.
- Тази функция може да бъде извикана **най-много 2100 пъти** в рамките на един тест.

Тази функция следва да използва следната функция, за да изпрати пакет:

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

- $A$: вектор с дължина $31$, който представлява битовете, изпратени от Айша в оригинален пакет.
- Тази функция връща потенциално променен пакет $B$, който представлява битовете, които Басма ще получи.
- Тази функция може да бъде извикана най-много $100$ пъти в рамките на едно извикване на `send_message`.

Втората функция, която трябва да имплементирате, е следната:

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

- $R$: вектор, описващ потенциално променените пакети. Потенциално променените пакети произлизат от реалните пакети, изпратени от Вас в ролята на Айша за едно извикване на `send_message`, и се подават **в реда, в който са изпратени** от Айша. Всеки елемент на $R$ е вектор с дължина $31$, който представлява потенциално променен пакет.
- Тази функция трябва да върне вектор от $S$ на брой бита, който трябва да е равен на първоначалното съобщение $M$.

- Тази функция може да бъде извикана **множество пъти** в рамките на един тест и **точно веднъж** за всяко съответстващо извикване на `send_message`. **Редът на извикванията** на `receive_message` не е задължително да е точно същият, като реда на съответстващото извикване на `send_message`.

Обърнете внимание, че извикванията на функциите `send_message` са в **едно изпълнение на вашата програма**, а тези на `receive_message` са в рамките на **друго изпълнение на вашата програма**.

## Ограничения

- $1 \leq S \leq 1024$ (**това е дължината на съобщението**)
- $C$ има точно $31$ елемента, като $16$ от тях са равни на $0$ и $15$ от тях са равни на $1$.

## Подзадачи и оценяване

Ако за някой от тестовете, за някое извикване на функцията `send_packet` не спазвате описаните по-горе правила или върната стойност на някоя извикване на `receive_message` не е вярна (дължината или съобщението е грешна), то ще получите резултат $0$ за този тест.

В противен случай, нека означим с $Q$ максималния брой извиквания на функцията `send_packet` сред всички извиквания на `send_message` сред всички тестове на дадена подзадача. Също, нека $X$ е равно на:

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

Тогава, резултатите на подзадачите се смятат по следния начин:

| Подзадача | Резултат | Допълнителни ограничения |
|---|---|---|
| 1 | $10 \cdot X$ | $S \leq 64$ |
| 2 | $90 \cdot X$ | Няма. |

Обърнете внимание, че грейдърът може да е **адаптивен**. Това означава, че стойностите, които са върнати от `send_packet` може да зависят не само от подадените параметри, а също и от много други неща, включително от подадените параметри и върнатите стойности на по-ранните извиквания на тази функция и псевдо произволни числа, генерирани от грейдъра. В този смисъл грейдърът е **детерминистичен**, защото ако го изпълните два пъти с изпращане на едни и същи пакети, то потенциално променените пакети ще са едни и същи.

## Пример

Нека имаме следното извикване.

```
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])
```

Съобщението, което Айша се опитва да изпрати на Басма е $[0, 1, 1, 0]$. Битовете с индекси от $0$ до $15$ не могат да бъдат променени от Клеопатра, докато битовете с индекси от $16$ до $30$ могат да бъдат променени от Клеопатра.

За целта на примера, нека предположим, че Клеопатра променя последователните битове, които тя контролира, с редуваща се редица от $0$ и $1$, т.е. тя избира $0$ за първия индекс, който контролира (индекс $16$ в нашия случай), $1$ за втория индекс, който контролира (индекс $17$), $0$ за третия индекс, който контролира (индекс $18$), и т.н.

Айша може да избере два бита от първоначалното съобщение в някой пакет по следния начин: тя ще изпрати първия бит като го повтори в първите $8$ индекса, които Клеопатра не контролира (тези с индекси от $0$ до $7$), а втория бит, като го повтори в следващите $8$ индекса, които Клеопатра не контролира (тези с индекси от $8$ до $15$).

Айша изпраща следния 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, 0])
```

Обърнете внимание, че Клеопатра може да промени само последните $15$ бита. Това означава, че Айша може да ги сложи по произволен начин, защото те може да бъдат променени. Като имаме предвид предполагаемата стратегия на Клеопатра, функцията връща: $[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]$.

Айша избира да изпрати последните два бита на съобщението $M$ във втория пакет по подобен начин като първия:

```
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, 0])
```

Като имаме предвид предполагаемата стратегия на Клеопатра, функцията връща: $[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]$.

Айша може да изпрати още пакети, но тя избира да не го прави.

След това грейдърът прави следното извикване:

```
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 възстановява съобщението $M$ по следния начин. От всеки пакет тя избира първия бит, който се повтаря два пъти подред в редицата, и последния бит, който се повтаря два пъти подред. По този начин, от първия пакет тя избира битовете $[0, 1]$, а от втория пакет избира битовете $[1, 0]$. Като ги обединява, тя възстановява цялото първоначално съобщение $[0, 1, 1, 0]$, което е вярната стойност за връщане на това извикване на функцията `receive_message`.

Може да бъде показано, че за тази предполагаема стратегия на Клеопатра и за съобщенията с дължина $4$, този подход на Басма правилно възстановява съобщенията, независимо от стойностите в $C$. Разбира се, в общия случай това не е вярно.

## Локален грейдър

Локалният грейдър не е адаптивен. Вместо това, Клеопатра променя последователните битове, които тя контролира с алтерниращa редица от $0$ и $1$, както беше описано по-горе.

Входен формат: **Първият ред на входа съдържа цяло число $T$, задаващо броя сценарии.** Следват $T$ сценарии. Всеки от тях се описва в следния формат:

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

Изходен формат: Локалният грейдър записва резултата от всеки от $T$-те сценарии в същия ред, в който са въведени:

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

Тук, $K$ е броят извиквания на функцията `send_packet`, $D$ е съобщението, което е върнала функцията `receive_message` и $L$ е неговата дължина.