Wiadomość

Aisha i Basma to dwie przyjaciółki, które ze sobą korespondują. Aisha ma wiadomość M, będącą sekwencją S bitów (tj. zer i jedynek), którą chciałaby przekazać Basmie. Aisha komunikuje się z Basmą wysyłając jej **pakiety**. Pakiet to sekwencja 31 bitów indeksowanych od 0 do 30. Aisha chciałaby przekazać Basmie wiadomość M wysyłając jej pewną liczbę pakietów.

Niestety Kleopatra uniemożliwiła komunikację między Aiszą i Basmą, i jest w stanie **zanieczyścić** pakiety. Oznacza to, że w każdym pakiecie Kleopatra może modyfikować bity na dokładnie 15 indeksach. Konkretnie, dana jest tablica C o długości 31, w której każdy element to 0 albo 1, z poniższym znaczeniem:

- $C[i] = 1$ oznacza, że bit o indeksie i może zostać zmieniony przez Kleopatrę. Nazywamy te indeksy **kontrolowanymi** przez Kleopatrę.
- $C[i] = 0$ oznacza, że bit o indeksie i nie może zostać zmieniony przez Kleopatrę.

Tablica C zawiera dokładnie 15 jedynek i 16 zer. Podczas wysyłania wiadomości M zestaw indeksów kontrolowanych przez Kleopatrę pozostaje taki sam dla wszystkich pakietów. Aisha dokładnie wie, które z 15-tu indeksów są kontrolowane przez Kleopatrę. Basma wie tylko, że 15 indeksów jest kontrolowanych przez Kleopatrę, ale nie wie które to z nich.

Niech A będzie pakietem, który postanawia wysłać Aisha (nazwiemy go **oryginalnym pakietem**). Niech B będzie pakietem odebranym przez Basmę (nazwiemy go **zanieczyszczonym pakietem**). Dla każdego i takiego, że $0 \leq i < 31$:

- jeśli Kleopatra nie kontroluje i-tego bitu ($C[i] = 0$), Basma otrzymuje i-ty bit wysłany przez Aishę ($B[i] = A[i]$),
- w przeciwnym wypadku, jeśli Kleopatra kontroluje i-ty bit ($C[i] = 1$), wartość $B[i]$ ustalana jest przez Kleopatrę.

Od razu po wysłaniu każdego pakietu Aisha dowiaduje się, jak wygląda zanieczyszczony pakiet.

Gdy Aisha wyśle już wszystkie pakiety, Basma otrzymuje wszystkie zanieczyszczone pakiety **w kolejności, w jakiej zostały wysłane**, a następnie musi odtworzyć wiadomość M.

Twoim zadaniem jest opracowanie i wdrożenie strategii, która pozwoli Aiszy przekazać Basmie wiadomość M tak, aby Basma mogła odzyskać M na podstawie zanieczyszczonych pakietów. Konkretnie, należy zaimplementować dwie funkcje. Pierwsza funkcja wykonuje czynności Aiszy. Dostaje wiadomość M i tablicę C, a jej zadaniem jest wysłać pakiety tak, aby przekazać Basmie

wiadomość. Druga procedura wykonuje czynności Basmy. Dostaje zanieczyszczone pakiety, a jej zadaniem jest odzyskanie oryginalnej wiadomości M.

Szczegóły implementacji

Pierwszą funkcją, którą powinieneś zaimplementować jest:

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

- M: tablica o długości S, opisująca wiadomość, którą Aisza chce przekazać Basmie.
- C: tablica o długości 31, opisująca indeksy bitów kontrolowanych przez Kleopatrę.
- Ta procedura może zostać wywołana **co najwyżej 2100 razy** w każdym teście.

Ta funkcja powinna wywoływać poniższą funkcję aby wysłać pakiet:

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

- A: oryginalny pakiet (tablica o długości 31), opisujący bity wysłane przez Aishę.
- Ta funkcja zwraca zanieczyszczony pakiet B reprezentujący bity, które zostaną odebrane przez Basmę.
- Tę procedurę można wywołać maksymalnie 100 razy, w trakcie każdego wywołania `send_message`.

Drugą funkcją, którą powinieneś zaimplementować, jest:

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

- R: tablica opisująca zanieczyszczone pakiety. Pakiety pochodzą z pakietów wysłanych przez Aishę w jednym wywołaniu `send_message` i są podane **w kolejności, w jakiej zostały przesłane** przez Aishę. Każdy element R jest tablicą o długości 31 opisującą zanieczyszczony pakiet.
- Ta procedura powinna zwrócić tablicę S bitów, będących oryginalną wiadomością M.
- Ta procedura może zostać wywołana **wiele razy** w każdym teście, **dokładnie raz** dla każdego odpowiadającego wywołania `send_message`. **Kolejność** wywołań procedury `receive_message` nie jest koniecznie taka sama, jak kolejność odpowiadających jej wywołań `send_message`.

Należy pamiętać, że w systemie oceniania procedury `send_message` i `receive_message` są wywoływane w **dwóch osobnych programach**.

Ograniczenia

- $1 \leq S \leq 1024$

- C ma dokładnie 31 elementów, z których 16 jest równych 0, a 15 jest równych 1.

Podzadania i punktacja

Jeżeli w którymkolwiek z przypadków testowych wywołania procedury `send_packet` nie spełniają powyższych zasad, lub wartość zwracana przez dowolne wywołanie procedury `receive_message` jest niepoprawna, Twoje rozwiązanie otrzyma 0 punktów za cały test.

W przeciwnym wypadku, niech Q będzie maksymalną liczbą wywołań procedury `send_packet`, spośród wszystkich wywołań `send_message`, we wszystkich przypadkach testowych. Niech X będzie równe:

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

Wynik całego testu obliczany jest w następujący sposób:

Podzadanie	Wynik	Dodatkowe ograniczenia
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	brak dodatkowych ograniczeń

Należy pamiętać, że w niektórych przypadkach zachowanie programu oceniającego może być **adaptywne**. Oznacza to, że wartości zwrócone przez `send_packet` mogą zależeć nie tylko od argumentów wejściowych, ale także od wielu innych rzeczy. W tym argumentów wejściowych poprzednich wywołań funkcji oraz ich zwracanych wartości, a także liczb pseudolosowych generowanych przez program oceniający. Tym niemniej program oceniający jest **deterministyczny**, to znaczy dwa uruchomienia na tym samym pakiecie zawsze zmienią go w taki sam sposób.

Przykład

Rozważmy następujące wywołanie.

```
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])
```

Wiadomość, którą Aisha próbuje wysłać Basmie to $[0, 1, 1, 0]$. Bity o indeksach od 0 do 15 nie mogą być zmieniane przez Kleopatrę, podczas gdy bity o indeksach od 16 do 30 mogą być zmieniane przez Kleopatrę.

Dla przykładu, załóżmy, że Kleopatra wypełnia kolejne bity, które kontroluje, naprzemiennie 0 i 1, tzn. przypisuje 0 do pierwszego indeksu, który kontroluje (w naszym przypadku jest to indeks 16), 1 do drugiego indeksu, który kontroluje (indeks 17), 0 do trzeciego indeksu, który kontroluje (indeks 18), i tak dalej.

Aisha może wysłać dwa bity z oryginalnej wiadomości w jednym pakiecie w następujący sposób: wyśle pierwszy bit na pierwszych 8 indeksach, które kontroluje, a drugi bit na kolejnych 8 indeksach.

Aisha postanawia zatem wysłać następujący pakiet:

```
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])
```

Należy pamiętać, że Kleopatra może zmieniać tylko bity znajdujące się na ostatnich 15 indeksach, więc Aisha może ustawić je dowolnie, gdyż mogą zostać nadpisane. Przy założonej strategii Kleopatry funkcja zwraca: $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Aisha decyduje się wysłać ostatnie dwa bity M w drugim pakiecie, podobnie jak poprzednio:

```
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])
```

Przy założonej strategii Kleopatry funkcja zwraca: $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$.

Aisha może wysłać więcej pakietów, ale nie chce tego robić.

Następnie moduł oceniający wywołuje następującą funkcję:

```
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 odzyskuje wiadomość M w następujący sposób. Z każdego pakietu bierze pierwszy bit, który pojawia się dwa razy z rzędu, a także ostatni bit, który pojawia się dwa razy z rzędu. Oznacza to, że z pierwszego pakietu pobiera bity $[0,1]$, a z drugiego pakietu bity $[1,0]$. Łącząc je, odzyskuje wiadomość $[0,1,1,0]$, będącą poprawnym wynikiem dla tego wywołania `receive_message`.

Można wykazać, że przy założonej strategii Kleopatry i dla wiadomości o długości 4, takie podejście Basmy poprawnie odzyskuje M, niezależnie od wartości C. Jednak w ogólnym przypadku nie jest to prawdą.

Przykładowy program oceniający

Przykładowy program oceniający nie jest adaptywny. Zamiast tego Kleopatra wypełnia kolejne bity, które kontroluje, naprzemiennie 0 i 1 tak, jak opisano w przykładzie powyżej.

Format wejścia: **Pierwszy wiersz wejścia zawiera liczbę całkowitą T, określającą liczbę przypadków testowych.** Następnie znajduje się opis T przypadków testowych. Każdy z nich jest dostarczany w następującym formacie:

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

Format wyjścia: Program oceniający zapisuje wynik każdego z T scenariuszy, w takiej samej kolejności, w jakiej zostały podane na wejściu, w następującym formacie:

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

K to liczba wywołań `send_packet` , D to wiadomość zwrócona przez `receive_message`, a L jest jej długością.