


















































Meddelande

Aisha  och  Basma  är  två 2 kompisar  som $+$ kommunicerar  med  varandra .

Aisha  har  ett 1 meddelande  M , som $+$ B består av  en 1 sekvens  av  S bitar $0\ 1$ (det  vill  säga  ett 1 och  nollor 0), som $+$ hon  vill  skicka  till  Basma . Aisha  kommunicerar  med  Basma  genom  att 2 skicka  **paket**  till  henne . Ett 1 paket  är  en 1 sekvens  av  31 bitar $0\ 1$, indexerade 0 från  0 till  30 . Aisha  vill  skicka  meddelandet  M till  Basma  genom  att 2 skicka  henne  ett 1 flertal  paket .

Tyvärr så har den ondskefulla POclatra saboterat kommunikationen mellan Aisha och Basma och kan **fördärva** paketen. Detta innebär att POclatra kan modifiera bitar i varje paket på exakt 15 index. Mer exakt så finns det en array C med längd 31, där varje element är antingen 0 eller 1, med följande betydelse:

- $C[i] = 1$ betyder att biten med index i kan ändras av POclatra. Vi säger att dessa index är **kontrollerade** av POclatra.
- $C[i] = 0$ betyder att biten med index i inte kan ändras av POclatra.

Arrayen C innehåller exakt 15 stycken ettor och 16 nollor. Indexarna som POclatra kontroller är samma för alla paket som skickas. Medan Aisha skickar meddelandet M vet hon exakt de 15 index som är kontrollerade av POclatra. Basma vet bara att 15 index är kontrollerade av POclatra, men hon vet inte vilka index.

Låt A vara ett paket som Aisha skickat (som vi kallar **originalpaketet**). Låt B vara paketet som Basma mottar (som vi kallar det **fördärvade paketet**). För varje i , så att $0 \leq i < 31$:

- om POclatra inte kontrollerar biten med index i ($C[i] = 0$), så mottar Basma biten i , precis som Aisha skickade den ($B[i] = A[i]$),
- annars, om POclatra kontrollerar biten med index i ($C[i] = 1$), så bestämmer POclatra värdet av $B[i]$.

Efter att ha skickat paketet får Aisha veta vad det motsvarande fördärvade paketet blev.

Efter att Aisha har skickat alla paketen, så mottar Basma alla fördärvade paket **i ordningen som de skickades**, och måste rekonstruera det ursprungliga meddelandet M .

Din uppgift är att komma på och implementera en strategi som låter Aisha skicka meddelandet M till Basma, så att Basma kan återskapa M från de fördärvade paketen. Mer exakt ska du

implementera två funktioner. Den första funktionen utför Aishas handlingar. Den får meddelandet M och arrayen C , och ska skicka ett flertal paket för att överföra meddelandet till Basma. Den andra funktionen utför Basmas handlingar. Den får de fördärvade paketen och ska återskapa det ursprungliga meddelandet M .

Implementationsdetaljer

Den första funktionen du ska implementera är:

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

- M : en array av längd S , som beskriver meddelandet som Aisha ska skicka till Basma.
- C : en array av längd 31 som beskriver de index som kontrolleras av POclatra.
- Denna funktion kommer att anropas **som mest 2100 gånger** per testfall.

Denna funktionen ska anropa följande funktion för att skicka ett paket:

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

- A : ett originalpaket (en array av längd 31) som beskriver bitarna som Aisha skickar.
- Denna funktionen returnerar ett fördärvat paket B som beskriver bitarna som kommer att mottas av Basma.
- Denna funktionen får anropas som mest 100 gånger per anrop av send_message.

Den andra funktionen du ska implementera är:

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

- R : en array som beskriver de fördärvade paketen. Paketen kommer att ha skapats genom att Aisha anropade send_message, och kommer att ges **i ordningen som de skickades** av Aisha. Varje element i R är en array av längd 31, som beskriver ett fördärvat paket.
- Denna funktionen ska returnera en array med S bitar, som ska vara samma som det ursprungliga meddelandet M .
- Denna funktionen kan bli anropad **flera gånger** i varje testfall, **exakt en gång** för varje motsvarande send_message-anrop. **Ordningen** receive_message-anrop är inte nödvändigtvis samma som ordningen av motsvarande send_message-anrop.

Notera att i domaren kommer send_message och receive_message-funktionerna att anropas i **olika program**.

Constraints

- $1 \leq S \leq 1024$

- C har exakt 31 element, varav 16 stycken är lika med 0 och 15 stycken är lika med 1.

Delpoäng och poängsättning

För varje testfall, om anropen till `send_packet` inte följer reglerna ovan, eller returvärdet av någon av anropen till `receive_message` är inkorrekta, så blir poängen på den gruppen 0.

Annars, låt Q vara det största antalet anrop till funktionen `send_packet` bland alla anrop av `send_message` över alla testfall. Låt också X vara:

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

Isåfall beräknas poängen som följande:

Grupp	Poäng	Ytterligare begränsningar
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Inga ytterligare begränsningar.

Notera att i vissa fall är beteendet av gradern **adaptiv**. Detta innebär att värdena som returneras av `send_packet` kan bero på värdet den anropas med, tillsammans med returvärden från tidigare anrop till denna funktionen.

Exempel

Betrakta följande anrop:

```
send_message([0, 1, 1, 0],
             [0, 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])
```

Meddelandet som Aisha försöker skicka till Basma är $[0, 1, 1, 0]$. Bitarna med index från 0 till 15 kan inte förändras av POclatra, medan bitarna med index från 16 till 30 kan förändras av POclatra.

För just det här exemplet, låt oss anta att POclatras använder en deterministisk strategi, och hon fyller intilliggande bitar som hon har kontroll över med varannan 0 och varannan 1. Det vill säga, hon fyller i 0 till första indexet som hon har kontroll över (index 16 i vårt fall), 1 till det andra indexet som hon kontrollerar (index 17), 0 det tredje indexet (index 18), och så vidare.

Aisha kan bestämma sig att skicka två bitar från det ursprungliga meddelandet i ett paket genom att göra följande: hon skickar första biten i de första 8 index som hon kontrollerar och den andra

biten vid de följande 8 index som hon kontrollerar.

Aisha bestämmer sig sedan att skicka följande paketet:

```
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])
```

Notera att POclatra kan ändra bitar bland de sista 15 index, så Aisha kan sätta de till vad som helst, eftersom det kan bli överskrivna. Om vi antar att POclatra använder den deterministiska strategin, så returnerar funktionen: $[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]$.

Aisha bestämmer sig för att skicka de två sista bitarna av M i det andra paketet på ett liknande sätt som förut:

```
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])
```

Om vi ännu en gång antar att POclatra använder samma deterministiska strategi, så returnerar funktionen: $[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]$.

Aisha kan skicka fler paket, men hon känner sig färdig.

Gradern gör sedan följande anrop:

```
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, 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]])
```

Basma återskapar meddelandet M genom att göra följande. För varje paket tar hon den första biten som dyker upp två gånger på raken, och den sista biten som dyker upp två gånger på raken. Det vill säga, från första paketet avläser hon bitarna $[0, 1]$, och från det andra paketet avläser hon bitarna $[1, 0]$. Genom att lägga ihop dessa värden, så kan hon återskapa meddelandet $[0, 1, 1, 0]$, vilket är det korrekta returvärdet för det här anropet till `receive_message`.

Det kan bevisas att om POclatra använder den ovannämnda deterministiska strategin och meddelandet har längd 4, så kan Basma återskapa M , oavsett vad C är. Detta funkar inte i allmänna fallet.

Exempel-Grader

Exempelgradern är inte adaptiv. Istället är POclatras beteende deterministiskt, och hon fyller intelligande bitar som hon kontrollerar med varannan 0 och varannan 1-bitar, som beskrivs i

exemplet ovanför.

Input-format: **Första raden av indata innehåller heltalet T , som beskriver antalet oberoende scenarion.** T scenarion följer. Varje scenario beskrivs i följande format:

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

Utdata-format: Exempelgradern skriver ut resultatet av varje av de T scenariorna i samma ordning som de ges i indata i följande format:

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

Här är K antalet anrop till `send_packet`, D är meddelandet som returnerades av `receive_message` och L är dess längd.