Upravené správy

Alica chce poslať svojmu kamarátovi Makerovi správu M. Jej správa je postupnosť S bitov (núl alebo jednotiek). Keďže sú informatici, Alica zásadne komunikuje s Makerom len tak, že mu pošle postupnosť **paketov**. Každý paket je postupnosť 31 bitov, ktoré sú očíslované od 0 po 30. Na poslanie správy M chce Alica využiť niekoľko paketov.

Ľadová kráľovná Lucka sa však rozhodla, že si z nich vystrelí a obsah paketov im upraví. Našťastie, pakety nemôže meniť ľubovoľne. Je obmedzená poľom C dĺžky 31, ktoré určuje, bity na ktorých pozíciách v pakete môže Lucka upraviť. Presnejšie, pole C obsahuje iba hodnoty 0 a 1, pričom

- ak $C[i] = 1$, tak bit na pozícii i je kontrolovaný Luckou a tá môže bit na tejto pozícii ľubovoľne zmeniť
- ak $C[i] = 0$, tak bit na pozícii i Lucka nemôže za žiadnych okolností zmeniť

Pole C obsahuje **práve** 15 **jednotiek** a 16 núl. To isté pole C je pre Lucku záväzné pre všetky pakety, ktoré Alica posiela Makerovi v rámci jednej správy M.

Alica pozná hodnoty poľa C, vie teda, ktoré bity môže Lucka potenciálne zmeniť. Maker však hodnoty poľa C **nepozná**, vie iba, že je v ňom presne 15 jednotiek.

Nech A je paket, ktorý chce Alica poslať Makerovi (nazvime ho **originálny paket**) a B je paket, ktorý dostane Maker po úprave Luckou (nazvime ho **upravený paket**). Pre každé $0 \leq i < 31$ platí:

- ak je $C[i] = 0$, tak $B[i] = A[i]$ – bit nemohol byť zmenený
- ak je $C[i] = 1$, hodnotu $B[i]$ nastavila Lucka ako sa jej páčilo

Hneď ako Alica odošle paket A, dozvie sa, ako vyzerá paket B – vidí teda, ako Lucka odoslaný paket upravila.

Keď Alica postupne vytvorí a odošle všetky pakety k správe M, príde na rad Maker. Ten dostane naraz všetky upravené pakety, a to **v rovnakom poradí, ako boli odoslané**. Musí pomocou nich zistiť obsah pôvodnej správy M.

Vašou úlohou bude vymyslieť a implementovať stratégiu, pomocou ktorej Alica odošle pakety a tiež stratégiu, pomocou ktorej Maker z upravených paketov zistí pôvodnú správu M. Vaše riešenie implementuje dve funkcie – prvá simuluje Alicu a na základe správy M a poľa C vytvorí sadu originálnych paketov, ktoré chce poslať Makerovi; druhá simuluje Makera, ktorý z obdržaných upravených paketov zrekonštruuje správu M.

Implementačné detaily

Prvá funkcia, ktorú potrebujete implementovať je funkcia:

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

- M: pole dĺžky S obsahujúce správu, ktorú chce Alica poslať Makerovi.
- C: pole dĺžky 31 určujúce pozície bitov, ktoré môže Lucka upravovať.
- Táto funkcia môže byť v rámci jednej testovacej sady zavolaná **nanajvýš 2100-krát**.

Pre poslanie paketu je v rámci funkcie `send_message` potrebné zavolať funkciu:

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

- A: originálny paket (pole boolov dĺžky 31), ktorý chce Alica poslať Makerovi.
- Táto funkcia vracia na výstup Luckou upravený paket B, ktorý bude poslaný Makerovi.
- Táto funkcia môže byť v rámci jednej funkcie `send_message` zavolaná **najviac 100-krát**

Druhá funkcia, ktorú potrebujete implementovať je funkcia:

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

- R: pole obsahujúce všetky upravené pakety. Toto sú všetky pakety, ktoré poslala Alica v rámci jedného volania funkcie `send_message` **v rovnakom poradí** ako boli poslané. Každý prvok poľa R je jedno pole dĺžky 31 popisujúce upravený paket.
- Táto funkcia má na výstup vrátiť pole S bitov, ktoré zodpovedá pôvodnej správe M.
- Táto funkcia bude zavolaná v každej testovacej sade **práve raz** pre každé volanie funkcie `send_message`. Avšak, **poradie** volania `receive_message` **nemusí byť rovnaké** ako poradie volania funkcie `send_message`.

V testovacom systéme budú funkcie `send_message` a `receive_message` volané v rámci **dvoch samostatných programov**.

Obmedzenia

- pre dĺžku správy M platí $1 \leq S \leq 1024$
- pole C má presne 31 prvkov, pričom práve 15 z nich sú jednotky a práve 16 z nich sú nuly.

Bodovanie

Ak v rámci testovania na nejakom vstupe zavoláte funkciu `send_packet` s nesprávnymi parametrami alebo funkcia `receive_message` vráti nesprávnu hodnotu, skóre za tento vstup (a teda aj za celú jeho podúlohu) bude 0.

Skóre za korektne vyriešený vstup sa vypočíta nasledovne: Nech Q je najväčší počet volaní funkcie `send_packet` v rámci jedného volania funkcie `send_message` v rámci daného vstupu.

Hodnota X je následne rovná:

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

Výsledné skóre za dotyčný vstup je vypočítané nasledovne:

Podúloha	Skóre	Dodatočné obmedzenia
1	$10 \cdot X$	$S \leq 64$
2	$90 \cdot X$	Bez dodatočných obmedzení.

Testovač je **adaptívny**. To znamená, že v niektorých testoch môže spôsob, akým upraví posielaný paket, závisieť nielen na samotnom pakete, ale aj na mnohých iných veciach, vrátane predchádzajúcich volaní `send_packet`, ich návratových hodnôt a pseudonáhodných čísel vygenerovaných samotným testovačom.

Testovač je **deterministický**, čo znamená, že keď ho spustíte dvakrát na tom istom teste a oba razy skúsite poslať tú istú postupnosť paketov, urobí v nich tie isté zmeny.

Príklad

Uvažujme nasledujúce volanie funkcie `send_message`.

```
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])
```

Alica chce Makerovi poslať správu $[0, 1, 1, 0]$ a Lucka môže meniť bity na pozíciách 16 až 30 (vrátane).

Pre ilustráciu predpokladajme, že Lucka v každom pakete bity, ktoré kontroluje, vypĺňa nastriedačku hodnotami 0 a 1. Teda hodnotu 0 dá na prvý ňou kontrolovaný index (v našom prípade index 16), hodnotu 1 na druhý ňou kontrolovaný index (v našom prípade index 17), opäť 0 na tretí ňou kontrolovaný index (v našom prípade index 18), atď.

Na poslanie prvých dvoch bitov správy môže Alica využiť nasledovnú stratégiu: prvý bit správy zapíše na prvých 8 indexov, ktoré nekontroluje Lucka; druhý bit správy zapíše na ďalších 8 indexov, ktoré nekontroluje Lucka.

Alica teda pošle nasledovný paket:

```
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])
```

Keďže Lucka kontroluje posledných 15 indexov, Alica ich môže nastaviť na ľuvoboľnú hodnotu, keďže aj tak môžu byť ľubovoľne zmenené.

Uvažujúc vyššie popísanú Luckinu stratégiu, funkcia `send_packet` vráti hodnotu $[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Zvyšné dva bity správy pošle Alica rovnakým spôsobom volaním funkcie:

```
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])
```

Počítajúc s Luckinou stratégiou bude výstupom funkcie paket $[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]$.

Alica by mohla Makerovi poslať aj ďalšie pakety, rozhodne sa však nepokračovať.

Testovač preto niekedy vykoná nasledovné volanie

```
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]])
```

Na prečítanie správy M môže Maker napríklad hľadať v každom pakete prvý výskyt dvoch rovnakých bitov za sebou a posledný výskyt dvoch rovnakých bitov za sebou. Hodnota týchto rovnakých bitov mu určí dva bity pôvodnej správy. Z prvého paketu dostane dvojicu bitov $[0,1]$, z druhého paketu dvojicu bitov $[1,0]$. Keď ich spojí, dostane korektnú správu $[0,1,1,0]$, ktorú vráti ako výstup funkcie `receive_message`.

Dá sa dokázať, že uvedená stratégia Alice a Makera funguje pri danom Luckinom správaní na všetkých správach dĺžky 4 bez ohľadu na to, aká je hodnota C. Samozrejme, toto neplatí vo všeobecnom prípade.

Ukážkový testovač

Ukážkový testovač nie je adaptívny, len simuluje Luckino chovanie, ktoré sme si popísali vyššie – teda Luckou kontrolované bity nahrádza nastriedačku hodnotami 0 a 1.

Formát vstupu: **Prvý riadok vstupu obsahuje celé číslo T určujúce počet rôznych testov.** Nasleduje T testov, každý je popísaný správou M a poľom C v nasledovnom formáte:

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

Formát výstupu: Ukážkový testovač vypíše na výstup výsledok každého z T testov v rovnakom poradí, ako boli zadané na vstupe a to v nasledovnom formáte:

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

Pole D obsahuje výstup funkcie `receive_message`, číslo L je dĺžka tohto výstupu a číslo K je počet volaní funkcie `send_packet` v tomto teste.