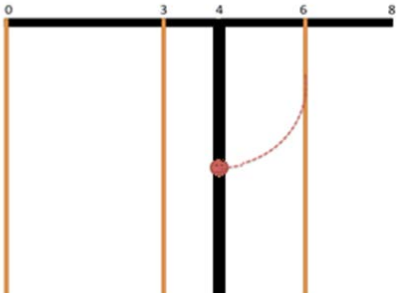
CIRCUS

Noul spectacol de la circ implică folosirea unor sfori foarte lungi ale căror capete sunt ataşate de tavanul plat al sălii. Intrarea în sală şi sforile sunt poziţionate în linie dreaptă. Tavanul se află la 10^{18} unităţi înălţime şi sforile sunt lăsate libere, fiind suficient de lungi pentru a atinge solul. Acrobaţii trebuie să se deplaseze de pe o sfoară pe alta astfel încât să ajungă la o distanţă de cel puţin M unităţi faţă de intrarea în sală.

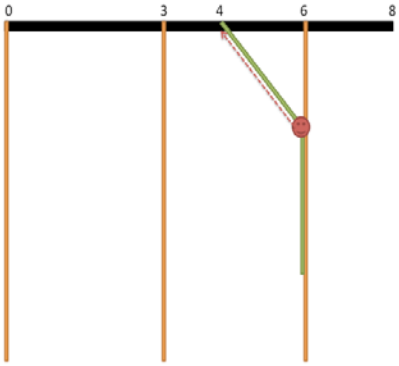
În total sunt N sfori. Sfoara cu numărul i este ataşată la o poziţie care se află la P_i unităţi de intrare, măsurate de-a lungul tavanului.

Acrobaţii sunt atenţi şi nu sar haotic de pe o sfoară pe alta. Imaginaţi-vă că un acrobat ţine sfoara cu numărul i la o distanţă de S unităţi de tavan. Acrobatul poate pendula pe sfoara pe care o ţine pentru a sări pe o altă sfoară. Dacă printr-o asemenea pendulare acrobatul poate atinge o poziţie la distanţă mai mare sau egală cu M faţă de intrare, putem considera că scopul său a fost îndeplinit. În termeni formali, acesta poate prinde sfoara cu numărul j dacă $|P_i - P_j| \leq S$. Acum acrobatul ţine atât sfoara cu numărul i cât şi sfoara cu numărul j. În continuare va urca pe sfoara cu numărul i ţinând încă în mână sfoara cu numărul j. Când acrobatul atinge punctul în care sfoara cu numărul i se ataşează de tavan are grijă ca sfoara cu numărul j să fie trasă strâns de-a lungul tavanului. Doar acum acrobatul poate continua mişcarea, rămânând pe sfoara cu numărul j la o distanţă de tavan egală cu lungimea cu care a fost ţinută sfoara pe suprafaţa tavanului. În termeni formali, acrobatul va continua mişcarea pe sfoara cu numărul j la o distanţă de $|P_i - P_j|$ de tavan.

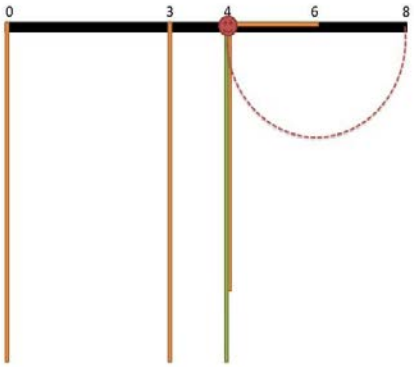
Administratorul circului doreşte să adauge încă o sfoară temporară ataşată la o poziţie care se află la D unităţi depărtare de intrare. Aceasta este sfoara pe care va începe spectacolul. Scopul acrobatului este să ajungă de pe acestă sfoară la o distanţă de cel puţin *M* unităţi faţă de intrare. În timpul mişcărilor de pe o sfoară pe alta nu se face nicio distincţie între sfoara temporară şi o sfoară obişnuită. Programul vostru trebuie să răspundă la întrebarea: care este distanţa minimă faţă de tavan la care acrobatul poate începe spectacolul pe sfoara temporară astfel încât să-şi poată îndeplini scopul.



Imaginaţi-vă un exemplu cu 3 sfori permanente poziţionate la 0, 3, respectiv 6 unităţi de intrare. Scopul este ca acrobatul să atingă punctul *M = 8*.



Imaginaţi-vă de-asemenea o sfoară temporară aflată la 4 unităţi de intrare şi un acrobat care se află pe sfoara temporară (cea îngroşată) la o distanţă de 3 unităţi de tavan. Acrobatul poate ajunge la sfoara care se află la 6 unităţi de intrare pendulând.



Odată ce acrobatul prinde sfoara care se află la 6 unităţi de intrare, începe să urce pe prima sfoară (cea temporară).

După ce urcă pe prima sfoară, acesta ţine acum a doua sfoară la o distanţă de 6 - 4 = 2 unităţi de baza acesteia. Acum acrobatul poate pendula de pe această sfoară şi ajunge la fix la ţinta care se află la 8 unităţi de intrare.

Cerinţă

Trebuie să implementaţi două funcţii: `init`, care este apelată o dată la începutul execuţiei programului şi care procesează parametrii iniţiali ai datelor de intrare şi `minLength`, care răspunde unei întrebări dându-se poziţia sforii temporare. Funcţia `minLength` va fi apelată de mai multe ori de către evaluator.

- `init(N, M, P[])`
  - N: numărul de sfori
  - M: distanţa ţintă în unităţi
  - P[]: un şir de lungime *N* care reţine poziţiile tuturor sforilor, măsurate în unităţi faţă de intare.

- `minLength(D)`
  - D: distanţa în unităţi faţă de intrare la care va fi ataşată sfoara temporară.

Detalii de implementare

Trebuie să trimiteţi spre evaluare exact un fişier, numit `circus.cpp`. Acest fişier implementază funcţiile `init` şi `minLength` folosind următoarele prototipuri.

- `void init(int N, int M, int P[])`
- `int minLength(int D)`

Fişierul **circus.cpp** NU trebuie să conţină o funcţie *main()*, dar poate conţine alte declaraţii şi funcţii care sunt considerate necesare pentru funcţionarea funcţiilor *init* şi *minLength.* Programul vostru trebuie să conţină linia `#include "circus.h"` la începutul său.

Restricţii

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Exemplu

$N = 3$, $M = 8$, $P = \{0, 3, 6\}$

`minLength(4)` trebuie să întoarcă valoarea 2. Este posibil ca acrobatul să sară de pe sfoara temporară pe sfoara care se află la poziţia 6, ţinând-o la 2 unităţi de tavan şi apoi să ajungă la poziţia $M = 8$. Altă soluţie posibilă este ca acrobatul să sară de pe sfoara temporară pe sfoara aflată la poziţia 0, apoi pe cea de pe poziţia 3, apoi pe cea de pe poziţia 6 şi să ajungă la $M = 8$. Totuşi, această soluţie ar necesita ca acrobatul să înceapă spectacolul pe sfoara temporară la o distanţă de 4 unităţi de tavan.

`minLength(5)` trebuie să întoarcă valoarea 3. Acrobatul are voie să atingă ţinta direct de pe sfoara temporară.

Subtasks

subtask	Points	*N*	minLength calls	Note
1	11	$0 \le N \le 100\,000$	**1**	Un singur apel minLength cu D = 0
2	17	$0 \le N \le 100$	**100**	$M \le 2000$
3	23	$0 \le N \le 2\,000$	**1 000**	
4	9	$0 \le N \le 2\,000$	**1 000 000**	
5	31	$0 \le N \le 100\,000$	**1 000**	
6	9	$0 \le N \le 100\,000$	**1 000 000**	

Testare locală

Pentru a vă putea testa funcţiile *init* şi *minLength* pe calculatorul vostru, veţi primi fişierele *Lgrader.cpp* şi *circus.h*. Compilaţi-le împreună cu fişierul **circus.cpp** şi veţi obţine un program pe care îl puteţi folosi pentru a vă testa funcţiile.

Lgrader citeşte datele de intrare în următorul format:

- linia 1: două valori întregi *N* şi *M*
- liniile **2 + *i*** ($0 \le i \le N-1$): P_i
- linia *N* + 2: *Q* – numărul de apeluri ale funcţiei `minLength`
- liniile ***N* + 3 + *i*** ($0 \le i \le Q-1$): valorile *D* - parametrii pentru fiecare apel al funcţiei `minLength`.

Lgrader va afişa *Q* valori, câte o valoare pe linie, reprezentând valorile produse de apelurile la funcţia `minLength`.

Exemplu pentru testare locală

Input	Output
3 8 0 3 6 2 4 5	2 3

HAPPINESS

Sistemul monetar din X - land este puţin ciudat. Există bancnote de valori egale cu toate numerele întregi între 1 şi *M*. În magazinele din X - land există o regulă ciudată – clientul nu primeşte niciodată rest, dar nici nu poate lăsa bacşiş – cu alte cuvinte clientul trebuie să plătească întotdeauna valoarea exactă a cumpărăturilor sale. Dacă el nu are suma exactă pentru achiziţia sa, atunci nu poate cumpăra. Imaginaţi-vă câte neplăceri crează acest lucru clienţilor.

Niya este o fată din X – land. Ca orice altă persoană, ea luptă constant împotriva regulei descrise mai sus. Ea cunoaşte întotdeauna setul său de bancnote – să presupunem că acestea au valorile $a_1, a_2,\ldots,a_N$. Valorile tuturor bancnotelor sunt între 1 şi *M* şi ea poate avea mai mult de o bancnotă de fiecare tip. De asemenea, secvenţa de valori $a_1, a_2,\ldots,a_N$ nu este ordonată în niciun fel. Atunci când intră într-un magazin, Niya se simte fericită dacă poate cumpăra orice combinaţie de produse având preţul total egal cu orice număr între 1 şi suma totală a bancnotelor sale $a_1 + a_2 + ... + a_N$. În cazul în care cumpără, ea trebuie să ia în considerare întreaga sumă de bani, fără să facă calcule complicate despre ce poate cumpăra (sau nu) cu bancnotele sale.

Observaţie: Să presupunem că $a_1, a_2,\ldots\ldots,a_N$ sunt ordonate crescător. Notăm cu $S_i = 1+a_1+ a_2+\ldots\ldots+a_i$. Condiţia necesară şi suficientă pentru a reprezenta fiecare număr între 1 şi $a_1+ a_2+\ldots\ldots+a_N$ ca sumă de elemente din multisetul $a_1, a_2,\ldots\ldots,a_N$, este ca inegalitatea $S_i \geq a_{i+1}$ să fie adevărată pentru fiecare $i>1$ şi $a_1 = 1$.

Setul de bancnote al fetei se modifică evident după fiecare achiziţie şi de asemenea după fiecare salariu primit – motiv pentru care fericirea ei este variabilă. Puteţi ajuta fata prin realizarea unui program. Programul vostru va primi la intrare setul initial de banknote al lui Niya şi toate evenimentele care au loc – achiziţii sau salarii. Programul trebuie să determine dacă Niya este fericită la începutul şi sfârşitul fiecărui eveniment.

Trebuie remarcat că Niya se simte fericită şi atunci când nu are deloc bani – atunci ea abandonează cumpărăturile şi merge să alerge.

Cerinţă

Scrie funcţiile *init() şi is_happy(),* care vor fi compilate cu evaluatorul juriului. Aceste funcţii vor folosi la determinarea fericirii lui Niya la începutul şi la sfârşitul fiecărui eveniment. Aceste funcţii vor primi ca parametri setul iniţial de bancnote şi seturile de bancnote care vor fi şterse din set (la cumpărare), respectiv adăugate setului (la primirea salariului).

Detalii de implementare

Trebuie să trimiteţi sistemului de evaluare fişierul **happiness.cpp,** care conţine funcţiile:

bool init(int coinsCount, long long maxCoinSize, long long coins[]).

bool is_happy(int event, int coinsCount, long long coins[]).

Descrierea parametrilor:

coinsCount – numărul bancnotelor primite (setul de start sau salariul) sau eliminate (cumpărare).

maxCoinSize – valoarea maximă a unei bancnote.

coins[] – tablou, în care, în ordine aleatoare, sunt date valorile bancnotelor (indicii pornesc de la 0).

event –tipul evenimentului:

−1 – cumpărare;
1 – primire salariu.

Funcţia *init* este apelată o singură dată de evaluator la început, pentru a stabili setul de start al bancnotelor lui Niya, apoi evaluatorul apelează de Q ori funcţia *is_happy* cu evenimentul = -1 (cumpărare) sau cu evenimentul = 1 (salariu). După fiecare apel, funcţia trebuie să returneze *true*, dacă Niya este fericită cu setul curent de bancnote sau *false* în caz contrar.

Fişierul **happiness.cpp** NU trebuie să conţină funcţia *main()*, dar poate conţine alte declarări şi funcţii, necesare pentru funcţionarea corectă a funcţiilor *init* şi *is_happy. P*rogram vostru trebuie să conţină directiva `#include "happiness.h"` la început.

Restricţii

Să notăm cu *Nc* numărul de bancnote al lui Niya la un moment dat şi K - numărul bancnotelor folosite la cumpărare sau primite ca salariu. Avem:

$0 \le Nc \le 200\,000$
$0 \le Q \le 100\,000$
$1 \le M \le 10^{12}$
$1 \le K \le 5$

Se garantează că la orice apel al funcţiei *is_happy* cu evenimentul = -1 (cumpărare), setul de bancnote memorat în tabloul *coins*[] este un subset al setului curent de banknote.

Exemple

called function	event	coinsCount	maxCoinSize	coins[]	function returns
init		5	100	4 8 1 2 16	true
is_happy	−1(shopping)	2		2 8	false
is_happy	1(receiving wage)	3		7 9 2	true

Subtasks

Subtask	Points	Nc	M	Q
1	10	≤ 300	≤ 100	≤ 100
2	20	≤ 20000	$\le 10^{12}$	≤ 1000
3	30	≤200000	≤ 1000000	≤ 100000
4	40	≤ 200000	$\le 10^{12}$	≤ 100000

Testare locală

Pentru a testa funcţiile *init* şi *is_happy* pe calculatorul local, trebuie să folosiţi fişierele *lgrader.cpp* şi *happiness.h*. Compilaţi-le împreună cu fişierul vostru **happiness.cpp** şi veţi obţine un program ce poate fi folosit la testarea funcţiilor voastre.

Programul aşteaptă următorul format al datelor de intrare:

Numerele naturale N şi M sunt date pe prima linie – numărul initial de bancnote şi valoarea maximă a unei bancnote.

Pe a doua linie se află N numere positive, separate prin spaţii – valorile bancnotelor din setul initial.

Pe a treia linie se află numărul natural strict pozitiv Q – numărul evenimentelor.

Pe fiecare dintre următoarele Q linii este descries un eveniment – mai întâi valoarea ce desemnează evenimentul: -1 (cumpărare), 1 (primire salariu). După aceasta se dă numărul natural *K* – numărul bancnotelor care sunt scoase ori adăugate din/in setul lui Niya. Ultimele *K* numere întregi, separate prin spaţii, reprezintă valorile bancnotelor care sunt scoase ori adăugate.

Programul va afişa la ieşirea standard Q + 1 linii cu 0 sau 1 – "starea de fericire" pentru Niya la începutul şi sfârşitul fiecărui eveniment.

Exemplu pentru testarea locală

Input	Output
5 100 4 8 1 2 16 2 -1 2 2 8 1 3 7 9 2	1 0 1

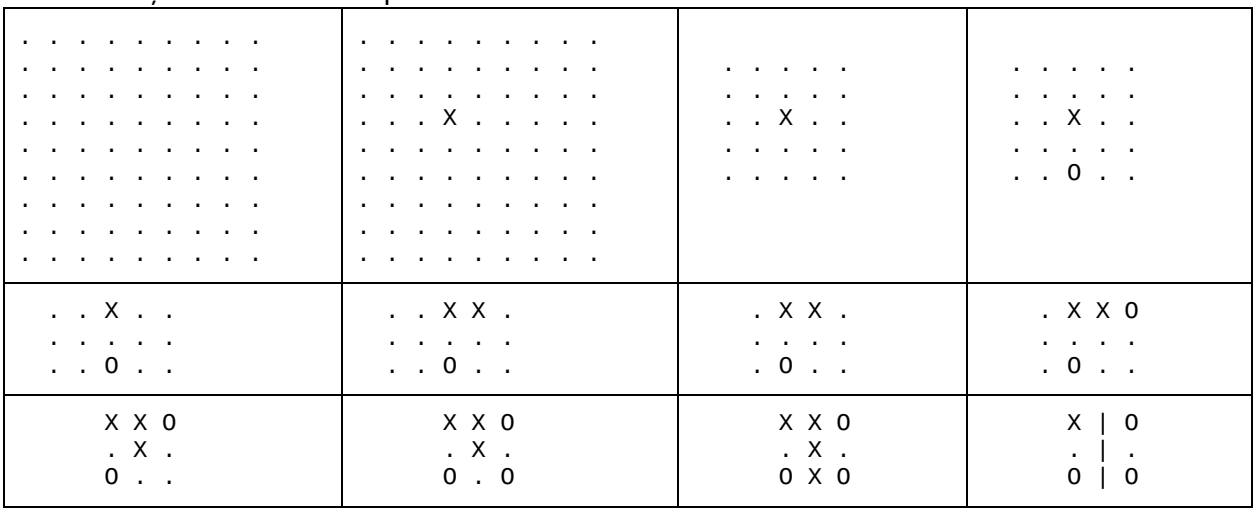
Ultimate TTT

Tic-Tac-Toe este un joc pentru două persoane cu reguli foarte simple. Jocul se joacă pe o tablă (inițial vidă) formată din trei linii și trei coloane. Primul jucător alege o celulă arbitrară și o completează cu simbolul său ('X'). După aceasta, al doilea jucător alege o celulă vidă și o completează cu simbolul său ('O'). La a treia mutare, primul jucător alege o altă celulă vidă și o completează cu simbolul său ('X') și așa mai departe. Jucătorii alternează mișcările și nici unul nu poate sări peste mișcarea sa (nu are nici un sens să sari peste o mișcare). Jocul continuă până când unul din jucători are trei din simbolurile sale aranjate pe o singura linie, coloană sau una din cele două diagonale. Dacă tabla s-a completat fără ca unul din jucători să aibă configurația câștigătoare, jocul se consideră remiză.

Elly și Kris joacă o versiune modificată a jocului. În loc de o tablă standard de 3 pe 3, ei încep jocul pe o tablă de dimensiuni infinite. După ce primul jucător face o mutare (nu are importanță unde), următoarea mutare poate fi făcută doar într-o celulă, care poate fi pe o tablă de 3x3 cu mutarea primului jucător. Adică, a doua mutare trebuie să fie la cel mult 2 coloane și/sau linii depărtare de celula marcată de primul jucător. Următoarele mutări urmăresc aceeași logică: jucătorii trebuie întodeauna să aleagă celula vidă asftel încât toate celule completate anterior pot forma o tabla de 3 pe 3. De notat că odată cu progresul jocului numărul de celule valide scade până când se obține o tablă de 3 pe 3 (se presupune că nici un jucător nu a invins până la acest moment). Vezi exemplul de mai jos pentru clarificări.

Ca și în jocul original, câștigător este jucătorul care completează primul celulele vecine de pe o linie, coloană sau diagonală. În mod similar, dacă toate celulele valide sunt completate înainte ca cineva să câștige, jocul este considerat remiză.

Considerați următorul exemplu:



În acest exemplu, primul jucător (‘X’) a câștigat. Dacă al doilea jucător ar fi folosit o strategie mai bună acest lucru nu s-ar fi întâmplat.

Vom numi *joc optimal* efectuarea acelor mutări care să permită jucătorului să câștige, daca jocul poate fi câștigat, să facă remiză dacă jocul nu poate fi câștigat dar poate fi o remiză sau poate fi pierdut în cazul în care nici o opțiune din cele de mai sus nu sunt satisfăcute. Se presupune că oponentul deasemenea joacă optim.

Cerință

Scrieţi un program care având o stare curentă a tablei, identifică celula care ar trebui aleasă de jucător dacă ar juca optimal.

Intrare

Prima linie din intrarea standard conține doi intregi **N** și **M** – numărul de linii și coloane rămase. Fiecare din următoarele **N** linii conține un șir de caractere de lungime **M**, care descrie o linie a tablei. Toate liniile formează o descriere corectă a stării curente a tablei. Este garantat că cel puțin o mutare a fost făcută (deci numărul de celule rămase este finit) și jocul încă nu a luat sfârșit.

Ieșire

Programul scrie pe o singură linie a ieșirii standard doi intregi **R** și **C** separați printr-un spațiu: respectiv numărul liniei și a coloanei (indexate începând cu 1), care indică celula pe care trebuie să o aleagă jucătorul pentru a juca optimal. În cazul în care sunt mai multe opțiuni, în ieșirea standard se poate indica oricare din ele.

Restricții

- $3 \leq \mathbf{N}, \mathbf{M} \leq 5$
- Toate simbolurile ce descriu tabla sunt din alfabetul {'.' (punct), 'X' (majusculă), 'O' (majusculă)}.
- Pentru 50% din punctaj testele au **N** = **M** = 3 (adică tabla corespunde jocului Tic-Tac-Toe original).

Notare

Testele sunt grupate în 4 subtaskuri. Fiecare subtask oferă 25 de puncte, dacă toate testele din acest subtask sunt trecute cu succes.

Exemplu

Intrare	Ieșire
3 4 .XX. .... .0..	1 1

Explicație

Este mutarea jucătorului al doilea (‘O’). Dacă se bifează celula (1, 4), atunci ‘X’ poate învinge bifând celula (2, 3) și punând-ul pe ‘O’ în “fork”. Nu contează ce celulă va alege ‘O’, ‘X’ va avea mutarea câștigătoare.

Bifând celula (1, 1), mutarea va limita tabla la coloanele 1..3, astfel încât va fi imposibil pentru ‘X’ să câștige dintr-o singură mișcare bifând celula (1, 4). Această mutare este o strategie care poate duce ‘O’ către o remiză.