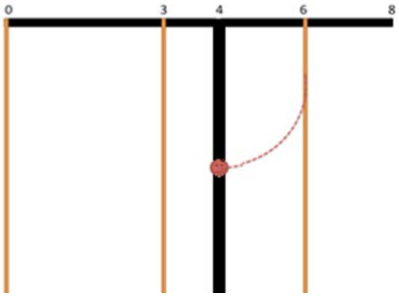
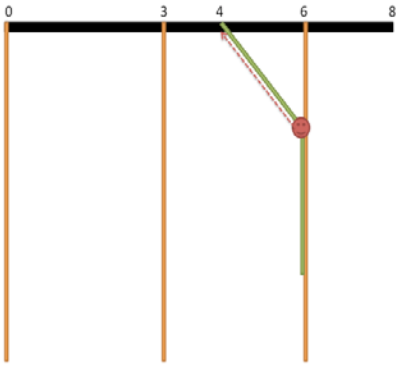
CIRCUS

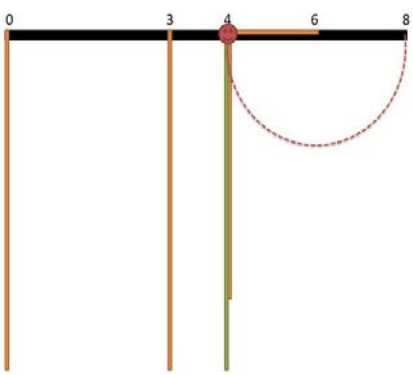
Novi cirkuski performans uključuje korišćenje jako dugačkih kanapa koji vise sa plafona cirkuske hale. Ulaz u cirkusku halu i pozicije svih kanapa nalaze se na jednoj pravoj liniji. Plafon je visok 10^{18} metara i svi kanapi su dovoljno dugački da dodiruju pod. Cirkusant mora skakati sa jednog kanapa na drug i cilj mu je da se udalji bar M metara od ulaza.

Ukupno ima N kanapa. i-ti kanap visi sa lokacije koja je udaljena P_i metara od ulaza, mjereno duž plafona.

Cirkusant je jako pažljiv i ne skače bezveze sa jednog kanapa na drugi. Zamislimo da se on drži za i-ti kanap na rastojanju S od plafona. On se može zaljuljati na kanapu koji drži. Ukoliko prilikom ljuljanja na kanapu dosegne tačku koja je na rastojanju ne manjem od M metara od ulaza, smatramo da je njegov zadatak ispunjen. Prilikom ljuljanja, on može zgrabiti bilo koji drugi kanap koji visi sa lokacije koja je udaljena najviše S metara od trenutnog kanapa. Formalnije, on može da zgrabi j-ti kanap ako je $|P_i - P_j| \leq S$. U tom trenutku (kada zgrabi j-ti kanap), on drži i i-ti i j-ti kanap. Tada on počinje da se penje duž i-tog kanapa, držeći sve vrijeme j-ti kanap. Kada dostigne tačku u kojoj i-ti kanap dodiruje plafon, on privuče j-ti kanap tako da bude potpuno zategnut duž plafona. Tek onda, on pušta i-ti kanap i prelazi na j-ti kanap i dolazi u situaciju da se nalazi na rastojanju od plafona koje je jednako dužini dijela j-tog kanapa koji je malopre privukao sebi. Formalnije, sada se on nalazi na j-tom konopcu na udaljenosti $|P_i - P_j|$ od plafona i može nastaviti sa kretanjem.







Direktor cirkusa želi da doda dodatni (privremeni) kanap i okači ga na plafon na rastojanju D metara od

ulaza, mjereno duž plafona. Na ovom kanapu će performans početi. Cirkusantov zadatak je i dalje da od ovog kanapa dođe do pozicije koja je udaljena bar *M* metara od ulaza. Prilikom prelaska sa kanapa na kanap, ovaj privremeni kanap se ne razlikuje od običnih (tj. važe ista pravila). **Vaš program mora da odgovori na pitanje: koje je najmanje rastojanje od plafona na kojoj on mora da se drži za ovaj privremeni kanap na početku performansa tako da može ispuniti svoj zadatak.**

Razmotrimo primjer sa 3 kanapa koja se nalaze na pozicijama 0, 3 i 6 metara od ulaza. Neka je cilj dostići *M* = 8.

Neka je dat privremeni kanap na rastojanju 4 metra od ulaza. Pretpostavimo da on drži privremeni kanap (boldiran na slici) na rastojanju 3 metra od plafona. On se može zaljuljati i uhvatiti za kanap koji je udaljen 6 metara od ulaza.

Čim zgrabi kanap koji je udaljen 6 metara od ulaza, on kreće da se penje uz trenutni kanap (u ovom slučaju – onaj privremeni)

Kada se popne do plafona, on drži drugi kanap u dužini 6 - 4 = 2 metara od njegove početne tačke. Sada može da pređe na drugi kanap, zaljulja se i dosegne traženu daljinu od 8 metara od ulaza.

Zadatak

Potrebno je implementirati dvije funkcije: `init`, koja se poziva samo jednom na početku vašeg programa i koja obrađuje početne ulazne parametre, i `minLength`, koja odgovara na gore opisani upit za datu lokaciju privremenog kanapa. Vaša funkcija `minLength` će biti pozivana nekoliko puta u toku izvršenja programa.

- `init(N, M, P[])`
  - `N`: broj kanapa
  - `M`: traženo rastojanje od ulaza (u metrima)

- `P[]`: niz dužine *N* u kome se nalaze lokacije svih kanapa, mjerene u metrima duž plafona (udaljenosti od ulaza).

- `minLength(D)`
  - D: rastojanje (u metrima) od ulaza, mjereno duž plafona, na kome će se nalaziti privremeni kanap u trenutnom upitu.

Detalji implementacije

Potrebno je da pošaljete tačno jedan fajl koji se mora zvati `circus.cpp`. Ovaj fajl mora implementirati `init` i `minLength` funkcije kao što je gore opisano, koristeći sljedeće potpise:

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

Fajl **circus.cpp** NE SMIJE sadržavati funkciju *main()*, ali može sadržavati dodatne promjenljive i funkcije ukoliko su potrebne za funkcije *init* i *minLength.* Vaš program mora sadržavati liniju `#include "circus.h"` na početku.

Ograničenja

$1 \leq M \leq 10^9$

$0 \leq P_i, D < M$

Primjer

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

Poziv funkcije `minLength(4)` treba da vrati 2. Moguće je preći sa privremenog kanapa na kanap na poziciji 6, držeći se za njega na rastojanju 2 metra od plafona a zatim ljuljanjem dostići rastojanje $M = 8$. Još jedan mogući način prelaska sa kanapa na kanap je preći sa privremenog kanapa na kanap na poziciji 0, zatim na kanap na poziciji 3, zatim na 6 a zatim dostići $M = 8$. Međutim, za ovaj niz poteza je neophodno da se na početku cirkusant drži za privremeni kanap na udaljenosti 4 metra od plafona.

Poziv funkcije `minLength(5)` treba da vrati 3. Primjetimo da on može dostići traženo rastojanje M odmah sa trenutnog kanapa.

Subtasks

subtask	Poeni	N	Broj poziva funkcije minLength	komentar
1	11	$0 \le N \le 100\,000$	**1**	Samo jedan poziv funkciji minLength sa parametrom 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**	

Local testing

Da biste mogli da testirate vaše funkcije *init* and *minLength* na vašem lokalnom računaru, dobićete fajlove *Lgrader.cpp* i *circus.h*. Kompajlirajte ih zajedno sa vašim fajlom **circus.cpp** i dobićete program koji možete koristiti za testiranje vaših funkcija.

Lgrader čita ulaz u sledećem formatu

- linija 1: dva cijela broja N i M
- linija $\mathbf{2 + i}$ ($0 \le i \le N$-1): P_i
- linija $N + 2$: Q – broj poziva funkciji `minLength`
- linija $\mathbf{N + 3 + i}$ ($0 \le i \le Q$-1): brojeve D – parametre za svaki poziv funkciji *`minLength`*

Lgrader će odštampati Q brojeva, po jedan u svakoj liniji, koji predstavljaju povratu vrijednost funkcije *`minLength`* za odgovarajuće pozive.

Primjer lokalnog testiranja

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

SREĆA

Monetarni sistem u X-zemlji je malo neobičan. Postoje novčanice s vrijednostima svih cijelih brojeva od 1 do *M*. Tu je i još jedno čudno pravilo - u prodavnicama X-zemlje - kupac nikada ne dobija kusur, ali ne može ostaviti ni napojnicu - drugim riječima kupac uvijek mora platiti tačnu vrijednost njegove kupovine. Ako nema tačan iznos za njegovu kupovinu, onda ne može ni kupiti. Zamislite samo kakva je to nevolja za kupce.

Niya je djevojka iz X-zemlje. Kao i mnoge druge osobe, ona se stalno bori protiv gore opisanih pravila. Ona uvijek zna sa kojim novčanicama raspolaže - pretpostavimo njihove vrijednosti su $a_1, a_2,\ldots,a_N$. Sve te vrijednosti su između 1 i *M* a ona naravno može imati više od jedne novčanice od svake vrste. Takođe redoslijed vrijednosti $a_1, a_2,\ldots\ldots,a_N$, nije određen na bilo koji način. Niya je srećna kad ulazi u prodavnicu, ona može kupiti bilo koju kombinaciju robe ukupne cijene koja je jednaka bilo kojoj vrijednosti između 1 i ukupnog zbira njenih novčanica $a_1 + a_2 + \ldots + a_N$.. U tom slučaju, kad kupuje, ona mora uzeti u obzir samo njen ukupni iznos novca bez komplikovanog računanja hoće li moći kupiti (ili ne) s novčanicama koje posjeduje.

Napomena: Neka je $a_1, a_2,\ldots\ldots,a_N$ sortiran u rastućem redoslijedu. Neka nam označava $S_i = 1+a_1+ a_2+\ldots\ldots+a_i$ Neophodan i dovoljan uslov da bi mogli da predstavljaju svaki broj između 1 i $a_1+ a_2+\ldots\ldots+a_N$ *kao zbir elemenata multiseta $a_1, a_2,\ldots\ldots,a_N$, da je sledeća nejednakost Si ≥ ai + 1 istinita za svaki i>1 i $a_1 = 1$.*

Naravno, Niyin skup novčanica se mijenja nakon svake kupovine i nakon svake plate koju primi – i to je razlog zašto je njezina sreća promjenjiva. Kao džentlmen pomozite djevojci programom. Vaš program će dobiti kao ulaz početni skup Niyinih novčanica i sve kupovine i plate. Program bi trebao da utvrditi da li je Niya srećnana na početku i nakon svake promjene.

Treba napomenuti da je Niya srećna i kad nema novca - onda preskače trgovinu i ide na džoging

Zadatak

Napiši funkciju *init()* i *is_happy()*, koja će biti kompajlirana graderom. Ove funkcije trebaju poslužiti za utvrđivanje Niyine sreću na početku i nakon svakog događaja. Funkcije će dobiti kao parametre početni set novčanica i skupove novčanica koje su uklonjeni iz skupa (prilikom kupovine) i dodate skupu (kod dobijanja plate)

Detalji implementacije

Trebalo bi da pošaljete grader sistemu fajl **happiness.cpp,** koji sadrži funkcije :

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

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

Parameters description:

coinsCount – broj novčanica koje su primljene (na startu ili od plate) ili date (kupovinom).

maxCoinSize – maximum vrijednosti jedne novčanice.

coins[] – array, u kojoj je slučajnim redom data vrijednost novčanica (index startuje od 0).

event – event's type :

−1 – Kupovina;
1 – Primanje plate.

Funkciju *init* poziva jednom grader na početku da bi postavio početni set novčanica koji Niya ima, a zatim grader poziva Q puta funkcija is_happy sa *event* = −1 (kupovina) ili *event* = 1 (plata). Nakon svakog poziva funkcija treba da vrati *true*, ako je Niya srećna sa svojim trenutnim setom novčanica ili *false*, ako nije.

File **happiness.cpp** NE SMIJE imati funkciju *main()*, ali može imati druge deklaracije i funkcije, neophodne za korektan rad funkcija *init* i *is_happy.* Tvoj program treba da sadrži #include "happiness.h" na početku.

OGRANIČENJA

Neka Nc označava broj novčanica koje Niya ima u bilo kojem trenutku a K - broj novčanica, koja se koristi prilokom kupovine ili plate. Tada imamo:

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

To je garancija da svaki poziv *happiness* sa *event* = −1 (kupovina) set novčanica dat u *coins*[] je podniz trenutnog Niyinog seta novčanica.

Primjer

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

Lokalno testiranje

Kako bi testirali svoje funkcije *init* i *is_happy* lokalno na kompjuteru, dobićete fajlove *lgrader.cpp* i *happiness.h*. Kompajlirajte ih zajedno sa vašim fajlom **happiness.cpp** i dobićete program da možete testirati svoje funkcije.

Program očekuje sledeći ulazni format:

Pojedinačni integers *N i M* su dati u prvom redu – inicijalni broj Niyinih novčanica i maksimalna vrijednost jedne novčanice.

N prirodni brojevi dati u drugom redu, odvojeni razmakom - vrijednosti novčanica u početnom setu.

Ne negativni prirodni broj *Q* je dat u trećem redu – event's count.

U svakom od sledećih *Q* redova jedan događaj je opisan – prvo, vrijednost za događaj je data: −1 (kupovina) ili 1 (primanje plate). Nakon toga pozitivni integer *K* – broj novčanica koje su date ili primljene Niyinom setu. Poslednjih *K* brojeva, razdvojeni razmacima – vrijednost novčanica koje su date ili primljene.

Na standardnom izlazu program će štampati *Q*+1 liniju sa 0 ili 1 – statusom Niyine "sreće" na početku I kraju svakog događaja:

Primjer lokalnog testiranja:

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 (Ixs Oks) je igra za dva igrača sa vrlo jednostavnim pravilima igre. Igra se odvija na (inicijalno praznoj) tabli sa tri reda I tri kolone. Prvi igrač bira jedno od polja i popunjava ga svojim znakom ("X"). Nakon toga drugi igrač bira jedno prazno polje i popunjava ga svojim znakom ("O" veliko slovo O). Treći potez igra prvi igrač ("X") birajući jedno prazno polje i tako dalje. Igrači igraju naizmjenično i ne mogu preskočiti svoj potez (to jednostavno nema smisla). Igra se nastavlja sve dok jedan od igrača ne postavi tri svoja znaka u istom redu, koloni, ili jednoj od dvije dijagonale. Ako je tabla popunjena, a nijedan od igrača nije postigao pobjedničku poziciju, igra je završila neriješeno.

Elly i Kris igraju modifikovanu verziju ove igre. Umjesto standardne table dimenzija 3x3, oni počinju na beskonačno velikoj tabli. Nakon što prvi igrač odigra potez (apsolutno je nebitno gdje), sljedeći potez može biti odigran samo na polju koje se potencijalno nalazi na tabli 3x3 zajedno sa prvim poljem. Drukčije rečeno, drugi potez mora biti najviše 2 kolone i/ili reda udaljen od prvog. Sljedeći potez slijedi sličnu logiku: igrač uvijek mora izabrati takvo prazno polje tako da već odigrani potezi mogu stati na tablu dimenzija 3x3. Obratite pažnju da što više igra odmiče, smanjuje se broj mogućih polja za igru, sve dok se ne formira standardna tabela 3x3 (pod pretpostavkom da nijedan igrač ne pobijedi prije toga). Pogledajte primjer koji slijedi da bi bolje razumjeli igru.

Kao u originalnoj igri, pobjednik je igrač koji prvi ima tri susjedna polja, u istom redu, istoj koloni, ili dijagonali. Slično, ako se sva validna polja popune prije nego što iko uspije pobijediti, smatra se da je igra završila neriješeno.

Pogledajte pažljivo sljedeći primjer:

. . . .	. . . . X	. . . . X	. . . . X . . . . 0 . .
. . X . . . . 0 . .	. . X X . . . . 0 . .	. X X . . . . . . 0 . .	. X X 0 . . . . . 0 . .
X X 0 . X . 0 . .	X X 0 . X . 0 . 0	X X 0 . X . 0 X 0	X | 0 . | . 0 | 0

U ovom primjeru prvi igrač (“X”) pobjeđuje, ali da je drugi igrač (“O”) koristio bolju strategiju to se ne bi desilo.

Smatrat ćemo *optimalnom igrom* igranje takvih poteza, tako da igrač pobjeđuje, ako ima mogućnost da pobijedi, igra neriješeno, ako nema mogućnost da pobijedi, odnosno može remizirati ili izgubiti, ako nijedna od prve dvije opcije nisu moguće. Podrazumijevamo da i drugi igrač igra optimalno.

Zadatak

Napišite program tako da ako je dato trenutno stanje na tabli treba da nađete koje polje igrač treba odabrati ako igra optimalno.

Ulaz

Prva linija standardnog ulaza sadrži dva cijela broja N i M – broj preostalih validnih redova(N) i broj preostalih validnih kolona(M). Svaka od sljedećih N linija sadrži string dužine M, koji opisuje taj red table. Sve linije formiraju ispravan opis trenutnog stanja na tabli. Garantuje se da je najmanje jedan potez odigran (tako da je preostali broj polja za igru konačan), i da igra još nije završena.

Izlaz

Program ispisuja u jednoj liniji standardnog izlaza dva cijela broja R i C (razdvojena jednim spejsom): gdje je R broj reda a C broj kolone (indeksirani počev od broja 1). Oni pokazuju polje koje sljedeći igrač treba odabrati tako da igra optimalno. Ako postoji više opcija, izlaz može da opiše bilo koju od njih.

Ograničenja

- $3 \leq \mathbf{N}, \mathbf{M} \leq 5$
- Svi simboli koji opisuju tablu su iz alfabeta {'.' (tačka), 'X' (veliko slovo), 'O' (veliko slovo)}.
- U testnim slučajevima, 50% vrijednosti poena za zadatak je za, $\mathbf{N} = \mathbf{M} = 3$ (znači tabla je standardna Iks Oks tabla 3x3).

Ocjenjivanje

Testiranja zadatka su grupisana u 4 podzadatka. Svaki podzadatak daje 25 poena, ako svi testovi u podzadatku budu ispravni.

Primjer

Ulaz

```
3 4
.XX.
....
.O..
```

Izlaz

```
1 1
```

Pojašnjenje

Na potezu je drugi igrač (“O”). Ako odabere (1,4), onda “X” može pobijediti igrajući (2,3) i stavljajući “O” u gubitničku poziciju bez obzira koje polje bira, tako da “X” ima pobjednički potez.

Igrajući na (1,1) sljedeći potez na tabli će biti limitiran na kolone 1..3, tako da će biti nemoguće za “X” da pobijedi u jednom potezu igrajući na (1,4). Koristeći ovaj potez imamo strategiju kojom “O” vodi igru u neriješenu poziciju.