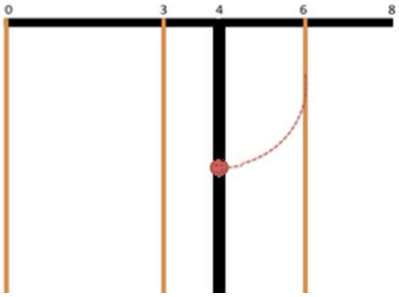
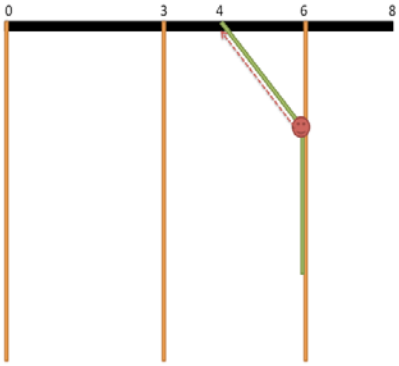
CIRKU

Një numër i ri cirku përdoron litarë (cima) me të vërtetë të gjatë, që varen nga tavani i rafshtë i sallës së cirkut. Hyrja në këtë sallë dhe të gjithë litarët janë pozicionuar në vijë të drejtë. Lartësia e tavanit është 10^{18} njësi dhe litarët varen lirisht dhe prekin token. Akrobatët duhet të lëvizin nga një litar në tjetrin ashtuqë të mund të mbërrijnë së paku M njësi larg hyrjes. Gjithsejt janë N litarë. Matur në tavanin e sallës, litari i i-të është i kapur në pozicionin që është P_i njësi nga hyrja.

Akrobatët janë të kujdesshëm dhe nuk kërcejnë marrëzisht nga një litar në tjetrin. Paramendoni një acrobat I cili është kapur për litarin e the i-të në distance prej S njësishë nga tavani. Akrobati mund të valëvisë në litarin që mbanë. Detyra e akrobatit konsiderohet e arrirë nëse duke u valëvitur në litar, akrobati mbërrinë pikën që nuk është më pak se M njësi nga hyrja. Duke valëvitur në litar, akrobati mund të kapet për një litar tjetër I cili është I kapur në një pike që është më së shumti S njësi nga litari i parlë. Thënë formalisht, akrobati mund të kapë litarin e j-të nëse $|P_i - P_j| \leq S$. Tani, (imagjinoni që) akrobati mbanë litarin e i-të dhe j-të. Në këtë moment, akrobati mund të ngjitet në litarin e i-të dhe njëkohësisht të mbajë në duar litarin e j-të. Kur akrobati mbërrinë pikën kur litari i i-të prek tavanin, ajo do të sigurohej që litari i j-të është i drejtuar shtërngueshëm me tavanin. Vetëm në këtë moment, akrobati mund të vazhdojë të lëvizë, duke mbajtur litarin e j-të, në distance nga tavani përcaktuar nga mënyra se si është shtërnguar litari përgjatë tavanit. Thënë formalisht, akrobati vazhdon të lëvizë duke u mbajtur në litarin e j-të në distance $|P_i - P_j|$ nga tavani.

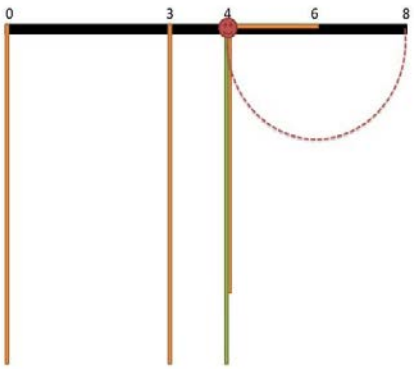
Shefi I cirkut dëshiron të shtojë një litar të përkohshëm, të pozicionuar në pikën që është *D* njësi nga hyrja (matur në tavan). Ky është litari ku numri fillon. Detyra e akrobatit është që, duke filluar nga ky litar, të mbërrisë në skajin tjetër të sallës: së paku *M* njësi nga hyrja. Gjatë lëvizjes nga një litar në tjetrin, litari I përkohshëm është I padallueshëm nga një litar I zakonshëm. Programi juaj duhet të japë përgjigje pyetjes: cila është distance minimale nga tavani, që akrobati duhet të mbajë litarin e përkohshëm, ashtuqë të plotësojë detyrën e vet.





Të marrim një shembull me 3 litarë permanentëtë pozicionuar respektivisht 0, 3 dhe 6 njësi nga hyrja. Qëllimi është që të arrihet *M* = 8.

Të marrim litarin e përkohshëm, 4 njësi nga hyrja. Imagjinoni akrobatin që mbanë litarin e përkohshëm (me vijën bold) në pikën që është 3 njësi nga tavani. Akrobati mund të arrisë litarin që është 6 njësi nga hyrja duke u lëkundur.



Pasiqë akrobati të ketë kapur litarin që është 6 njësi nga hyrja, ajo fillon të ngjitet lart në litarin e parë (këtu – litari i përkohshëm).

Pasi të jetë ngjitur në litarin e parë, ajo mbanë litarin e dytë 6 - 4 = 2 njësi nga pikëkapja e tij. Tani, akrobati është gati të lëkundet dhe të mbërrisë qëllimin që është 8 njësi larg hyrjes.

Task

Duhet të implementoni dy funksione: `init`, i cili thirret njëherë në fillimin e ekzekutimit të programit tuaj dhe proceson parametrat hyrës, dhe `minLength`, I cili jep përgjigje një pyetjeje (query) pasi të jetë dhënë vendndodhja e litarit të përkohshëm. Funksioni I juaj `minLength` do të thirret shumë here nga programi i vlerësimit (grading).

- `init(N, M, P[])`
  - N: numri I litarëve
  - M: distance që kërkohet dhënë në njësi
  - P[]: vargu me madhësi *N* që mbanë vendndodhjen e të gjithë litarëve, matur përgjatë tavanit me njësi nga hyrja
- `minLength(D)`
  - D: distanca në njësi përgjatë tavanit nga hyrja e litarit të përkohshëm.

Detajet e implementimit

Duhet të dërgoni **saktësisht një** fajl, I titulluar `circus.cpp`. Ky fajl implementon funksionet `init`, dhe `minLength` që I kemi përshkruar lart duke përdoru këto signatura.

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

Fajli **circus.cpp NUK** duhet të përmbajë funksionin *main()*, por mund të përmbajë deklarime dhe funksione të tjera, të nevojshme për punë korrekte të funksioneve *init* dhe *minLength.* Programi juaj **duhet** të përmbajë `#include "circus.h"` në fillim.

Kufizimet

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Shembull

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

`minLength(4)` duhet të kthejë 2. Është e mundshme që të kërcehet nga litari I përkohshëm në litarin në pozicionin 6, duke u kapur në të 2 njësi nga tavani dhe të arrihet $M = 8$. NJë sekuence tjetër e kërcimeve do të ishte të kërcehet nga litari I përkohshëm në litarin në pozicionin 0, pastaj 3, dhe pastaj 6, dhe të arrihet $M = 8$. Mirëpo, sekuenca e dytë e kërcimeve do të kërkonte që në fillim, akrobati të mbahet 4 njësi nga tavani në litarin e përkohshëm.

`minLength(5)` duhet të kthejë 3. Akrobati mund të arrisë daljen bile drejtpërdrejtë nga litari I përkohshëm.

Subtasks

subtask	Pikë	N	minLength calls	Shënim
1	11	$0 \le N \le 100\,000$	**1**	Vetëm një thirrje e minLength me 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**	

Testimi lokal

Në mënyrë që të testoni funksionet e juaja *init* dhe *minLength* në kompjuterat e juaj, do të ju jepet fajli *Lgrader.cpp* dhe *circus.h*. Kompajloni ata së bashku me fajlin tuaj **circus.cpp** dhe do të fitoni një program që mund ta përdorni për të testuar funksionet e juaja.

Lgrader lexon inputin në këtë format:

- rreshti i 1: dy numra të plotë *N* dhe *M*
- rreshti **2 + *i*** ($0 \le i \le N-1$): P_i
- rreshti $N + 2$: *Q* – numri sa here `minLength` do të thirret
- rreshti ***N* + 3 + *i*** ($0 \le i \le Q-1$): numrat *D* – parametrat për secilën thirrje të `minLength`

Lgrader do të printojë *Q* numra, një për rresht, duke dhënë vlerat që japin thirrjet e `minLength`.

Shembull për testin lokal

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

Lumturia

Sistemi monetar në X-land është një çikë i çuditshëm. Ka kartmonedhë që kanë vlerën e të gjithë numrave të plote nga 1 në *M*. Ekziston edhe nje rregull shumë I çuditshëm në dyqanet e X-land – klienti nuk merr kur kusur, por edhe nuk jep bakshish– me fjalë të tjera, klienti duhet gjithmonë të paguaj shumën egzakte të sendit që blenë. Nëse klienti nuk e ka shumën e mjaftuar për blerjen, atëherë ai nuk mund të blejë. Imagjinoni se çfarë vështirësie I shkakton kjo klientëve.

Niya është një vajzë nga X-land. Si të gjithë personat e tjerë, ajo tërë kohën lufton kundër rregullave të përshkruara më lart. Ajo gjithmonë e di listen e kartmonedhave të saja – supozojmë që vlerat e saja janë $a_1, a_2,\ldots,a_N$. Të gjitha këto vlera janë në mes të 1 dhe *M* dhe ajo mund të ketë disa kartmonedhe me të njëjtën vlerë. Gjithashtu, sekuenca e vlerave $a_1, a_2,\ldots\ldots,a_N$, nuk rradhitje të çfarëdoshme. Niya ndihet e lumtur, kur pasiqë të futet në dyqan, ajo mund të blejë çfarëdo kombinimi të sendeve me çmim total që është në mes numrit 1 dhe shumës totale të kartmonedhave të saja $a_1 + a_2 + ... + a_N$. Në këtë rast, gjatë blerjes, ajo duhet gjithmonë të ketë në konsideratë shumën totale të parave, pa bërë llogaritjet e komplikuara se a mund të blejë a jo me kartmonedhat e saja.

Vërejtje: Le ta sortojmë $a_1, a_2,\ldots\ldots,a_N$ në rradhitjen rritëse. Le të jetë $S_i = 1+a_1+ a_2+\ldots\ldots+a_i$. Kushti I nevojshëm dhe I domosdoshëm që një numër në mes të 1 dhe $a_1+ a_2+\ldots\ldots+a_N$ të përfaqësohet si shuma e elementeve nga sekuenca $a_1, a_2,\ldots\ldots,a_N$, është që jobarazimi $S_i \geq a_{i+1}$ të jetë i saktë për çdo *$i>1$ dhe $a_1 = 1$*.

Normalisht, lista e kartmonedhave të Niya's ndryshon pas çdo blerjeje dhe pas çdo rroge që merr – për këtë lumturia e saj është e ndryshueshme. Mund ti ndihmoni kësaj vajze me një program. Programi juaj do të marrë si input sekuencën iniciale të kartmonedhave të Niya's dhe të gjitha ngjarjeve që ndodhin – blerjet dhe pagat. Programi duhet të ketë mundësinë të përcaktojë nëse Niya është e lumtur në momentin fillestar dhe pas secilit event.

Duhet të cekim që Niya ndihet e lumtur edhe kur nuk ka para fare – atëherë ajo nuk del për të bërë Pazar por shkon për të vrapuar.

Task

Shkruaj funksionet *init()* dhe *is_happy()*, të cilët do të kompajlohen me graderin e jurisë. Këta funksione duhet të shërbejnë për të përcaktuar lumturinë e Niya's në momentin fillestar dhe pas çdo eventi. Këta funksione do të marrin si parametra sekuencën iniciale të kartmonedhave dhe sekuencën e kartmonedhave që janë hequr (në rastin e blerjes) ose shtuar në rastin e pagës.

Detajet e implementimit

Ju duhet të dorëzoni vetëm **një** file të quajtur **happiness.cpp,** i cili duhet të përmbajë këto funksione:

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

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

Përshkrimi I parametrave:

coinsCount – numri I kartmonedhave fillestare, të shtuara ose të hequra

maxCoinSize – vlera maksimale e një kartmonedhe.

coins[] – një array me integera në rradhitje të çfarëdoshme (numërimi filon nga 0)

event – tipi I eventit :

−1 – Blerje;
1 – Marrje rroge.

Funksioni *init* thërritet vetëm njëherë nga grader-I në fillim të programit për të përcaktuar monedhat që ka Niya në momentin fillestar. Grader-I më pas thërret funksionin is_happy *Q* herë me *event* = −1 (blerje) or *event* = 1 (marrje rroge). Për cdo thirrje, funksioni duhet të kthejë *true*, nëse Niya është e lumtur me listën e monedhave që ka pas ndryshimit të bërë, ose *false* nëse ajo nuk është.

File **happiness.cpp** NUK duhet të përmbajë funksion *main()*, por mund të përmbajë funksione dhe deklarime të tjera të domosdoshme për funksionimin e *init* dhe *is_happy.* Ky File duhet të përmbajë rreshtin `#include "happiness.h"` në filim.

Limitet

Lë të shënojmë me *Nc* numrin e kartmonedhave të Niyës në cdo moment kohor dhe *K* – numri I kartmonedhave të shpenzuara ose te marra në cdo blerje ose marrje rroge. Atëhere kemi:

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

Është e garantuar që në cdo thirrje të *is_happy* me *event* = −1 (blerje) cdo element në listën e kartmonedhave të dhëna në *coins*[] ndodhet në listën aktuale të kartmonedhave të Niyës.

Shembull

called function	event	coinsCount	maxCoinSize	coins[]	function returns
init		5	100	4 8 1 2 16	true
is_happy	−1(blerje)	2		2 8	false
is_happy	1(marrje rroge)	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

Testimi lokal

Për të testuar funksionet *init* dhe *is_happy* në kompjuterin tuaj, merrni file-t *lgrader.cpp* dhe *happiness.h*. Vendosini në një dosje dhe shtonjini në project bashkë me **happiness.cpp** dhe pasi t'i kompiloni do të merrni një program që teston funksionin tuaj.

Programi pret këtë format input-i:

Numra integer pozitivë *N* dhe *M* janë dhënë në rreshtin e parë – numri fillestar i kartmonedhave të Niyës dhe vlera maksimale e një kartmonedhe.

N numra integer pozitivë jepen në rreshtin e dytë, të ndarë me hapësira – vlerat e kartmonedhave në castin fillestar.

Numër integer jo-negativ *Q* jepet në rreshtin e tretë– numri i eventeve.

Në secilin prej Q rreshtave në vazhdim përshkruhet secili event – fillimisht jepet një vlerë për këtë event: −1 (blerje) ose 1 (marrje rroge). Më pas jepet një numer integer pozitiv – numri I kartmonedhave që janë shtuar ose hequr nga lista e Niyës. Në fund jepen K numra integer të ndarë me hapësira – vlera për secilën kartmonedhë që shtohet ose hiqet.

Nxirrni në ekran *Q*+1 rreshta me 0 ose 1 – statusi I "lumturisë" së Niyës në castin fillestar dhe pas çdo eventi

Shembull për testim lokal

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

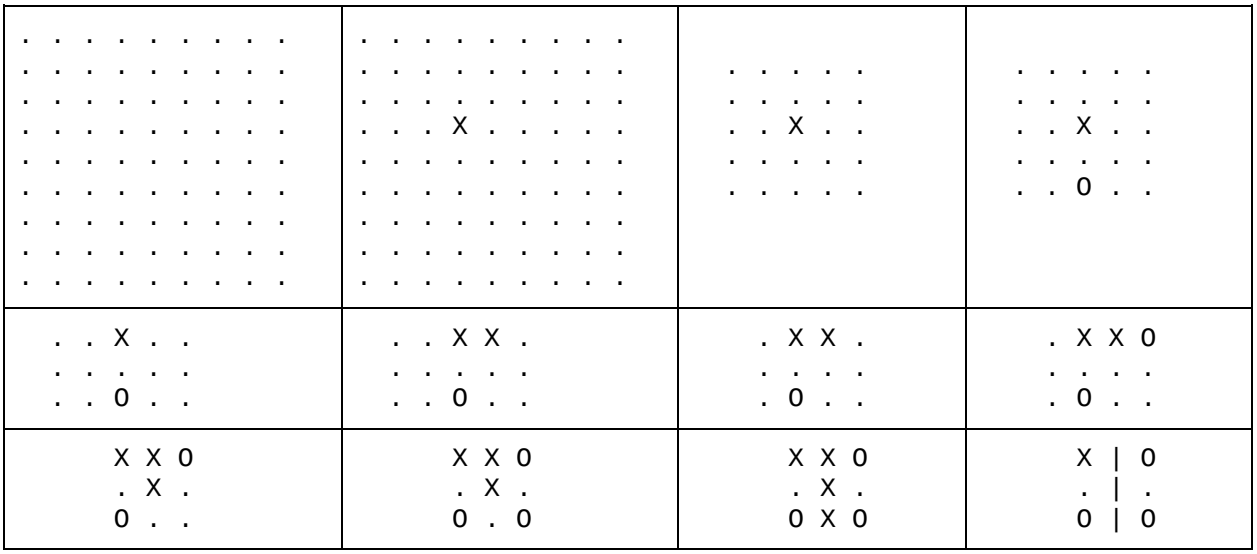
Ultimate TTT

Tic-Tac-Toe është një lojë e thjeshtë për dy persona. Loja luhet në një board (I cili është fillimisht bosh) me tre rreshta dhe tre kolona. Lojtari I parë zgjedh një qelizë dhe e mbush me shenjën e tij ('X'). Më pas, lojtari I dytë zgjedh një qelizë bosh dhe e mbush me shenjën e tij ('O'). Në raundin e dytë, lojtari I parë ('X') zgjedh përsëri një qelizë bosh dhe vendos shenjën e tij, e kështu me rradhë. Lojtarët luajnë me rradhë, njëri pas tjetrit, dhe nuk lejohet që të mos luajnë kur u vjen rradha e tyre (nuk ka kuptim loja në këtë mënyrë). Loja vazhdon në këtë mënyrë derisa njëri prej lojëtarëve të ketë në një prej rreshtave, kolonave ose diagonaleve tre shenjat e tij. Nëse boardi mbushet plot, por asnjëri prej lojëtarëve nuk ka mundur ta plotësojë me konfigurimin fitues, atëherë loja konsiderohet barazim.

Eli dhe Kris po luajnë një version të modifikuar të lojës. Në vend që të luajnë në një board standard 3x3, ata fillojnë të luajnë në një board me madhësi të pafundme. Pasi lojtari I parë bën një lëvizje (s'ka rëndësi se ku), lëvizja e dytë mund të bëhet vetëm në një prej qelizave që mund t'I përkiste të njëjtit board 3x3 me lëvizjen e parë. Me pak fjalë, lëvizja e dytë mund të bëhet vetëm në një qelizë me distancë maksimale 2 qeliza/rreshta nga e para. Levizjet e ardhshme ndjekin të njëjtën logjikë: lojtari mund të zgjedhë një qelizë bosh e cila, së bashku me të gjitha lëvizjet e mëparshme, mund të përmbahet në një board 3x3. Duhet vënë re se sa më shumë të vazhdojë loja, aq më shumë do të ulet numri I qelizave që mund të zgjidhen (përvec rastit kur një prej lojëtarëve fiton përpara se kjo të ndodhë). Shihni shembullin më poshtë për më shumë sqarime.

Njëlloj si loja origjinale, fitues quhet lojtari I cili arrin të vendosë shenjat e tij në të njëjtin rresht/kolonë ose diagonale. Gjithashtu, nëse boardi mbushet plot dhe asnjë prej lojëtarëve s'ka fituar ende, loja konsiderohet barazim.

Shihni shembullin në vazhdim:



Në këtë shembull lojtari ('X') fitoi, por nëse lojtari ('O') do të kishte luajtur me një strategji më të mirë, nuk do të ndodhte e njëjta gjë.

Themi se lojtari luan në mënyrë optimale atëherë kur lojtari fiton, nëse ekzison mundësia e fitores, del barazim nëse loja nuk mund të fitohet, por mund të arrihet që të dalë barazim, ose humbet nesë asnjë prej mundësive të mëparshme nuk mund te realizohet. Supozojmë që edhe kundershtari I tij luan në mënyrë optimale.

Task

Shkruani një program që, duke ditur gjendjen aktuale të boardit, gjen se cilën qelizë do të zgjidhte lojtari nëse ai do të luante në mënyrë optimale.

Input

Rreshti i parë përmban dy numra të plotë **N** dhe **M** – numri i rreshtave dhe i kolonave të vlefshme që kanë mbetur. Secili prej **N** rreshtave përmban një string me gjatësi **M**, i cili përshkruan këtë rresht në board. Të gjitha rreshtat sëbashku formojnë një përshkrim të gjëndjes aktuale të boardit. Është e garantuar që është bërë të paktën një lëvizje(kemi një numër të fundëm qelizash të mbetura) dhe se loja nuk ka mbaruar akoma.

Output

Programi duhet të shkruajë në një rresht të vetëm dy integers **R** dhe **C**: respektivisht një numër rreshti dhe kolone (numërimi fillon nga 1). Ato janë koordinatat e qelizës që do zgjedhë lojtari nëse ai do luajë në mënyrë optimale. Nëse ka disa mundësi, programi mjafton të përshkruajë një prej tyre.

Limitet

- $3 \le \mathbf{N}, \mathbf{M} \le 5$
- Të gjitha simbolet që përshkruajë boardin janë marrë nga alfabeti {'.' (pikë), 'X' (shkronjë e madhe e shtypit), 'O' (shkronjë e madhe e shtypit)}.
- Në disa test cases, të cilat sëbashku përbëjnë 50% të pikëve të ushrimit, **N** = **M** = 3 (d.m.th loja është thjeshtë një lojë standarte Tic-Tac-Toe).

Vlerësimi

Testet e problemit janë të ndara në katër grupe. Secili grup pëbën 25% të pikëve të cilat merren vetëm nësë programi i kalon të gjitha testet e këtij grupi.

Shembull

Input	Output
3 4 .XX. .... .O..	1 1

Shpjegimi

Është rradha e lojarit të dytë (‘O’). Nëse ai zgjedh qelizën (1, 4), atëhere ‘X’ fiton lojën duke e vendosur shenjën e tij në (2, 3) dhe duke e futur ‘O’ në një “kleckë” – s’ka rëndësi se clilën qelizë zgjedh ‘O’, ‘X’ do të fitojë gjithmonë.

Por duke zgjedhur (1, 1) atëhere loja do të limitohej vetëm në kolonat 1..3, duke e bërë të pamundur për ‘X’ të fitojë direkt duke zgjedhur (1, 4). Duke bërë këtë lëvizje, ekziston një strategji e cila bën që loja të dali barazim.