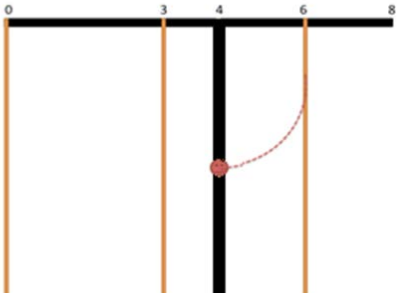
ЦИРКУС

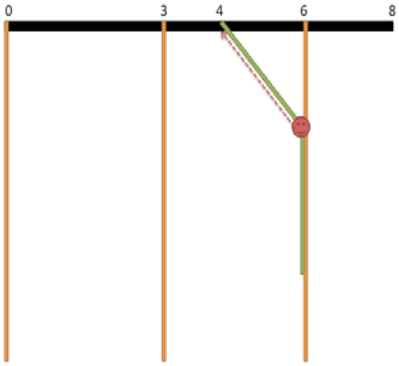
Еден нов циркуски перформанс вклучува користење на навистина долги јажиња, кои што се закачени на рамниот таван на циркуската сала, висејќи надолу. Влезот во циркуската сала и сите јажиња се поставени во права линија. Таванот е висок 10^{18} единици, а јажињата висат слободно и се протегаат до подот. Циркузантите мора да се придвижуваат од едно на друго јаже, за да можат да се оддалечат барем M единици од влезот.

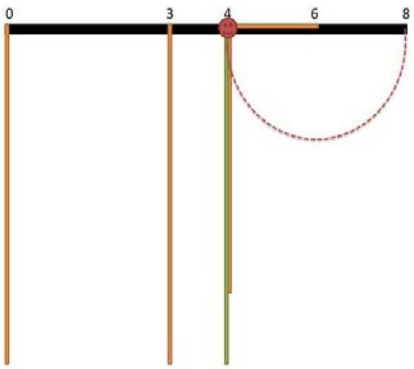
Има вкупно N јажиња. i-тото јаже е закачено на локација оддалечена P_i единици од влезот, мерено долж таванот на салата.

Циркузантите се внимателни и не скокаат неразумно од едно на друго јаже. Замислете дека еден циркузант е фатен за i-тото јаже на вертикално растојание од S единици од таванот. Циркузантот може да се ниша на јажето за кое што е фатен. Ако, нишајќи се на некое јаже, циркузантот достигне до точка која е оддалечена не помалку од M единици од влезот, може да сметаме дека неговата задача е завршена. Додека се ниша, циркузантот може да се фати и за друго јаже, кое што е закачено на локација оддалечена најмногу S единици од тековното јаже. Формално, циркузантот може да се фати за j-тото јаже ако $|P_i - P_j| \leq S$. Сега, циркузантот е фатен и за i-тото и за j-тото јаже. Во овој момент, тој почнува да се искачува по i-тото јаже, држејќи го во исто време во своите раце и j-тото јаже. Кога циркузантот ќе ја достигне точката во која што i-тото јаже го допира таванот, тој ќе се осигура дека j-тото јаже е цврсто закачено повлекувајќи го долж таванот. Само во ваков момент, циркузантот може да продолжи да се движи, држејќи се за j-тото јаже, на вертикално растојание од таванот базирано на тоа како јажето било врзано за таванот. Формално, циркузантот го продолжува своето движење држејќи се за j-тото јаже на растојание $|P_i - P_j|$ од таванот.

Менаџерот на циркусот сака да додаде дополнително, помошно јаже, кое ќе биде закачено на позиција која што е оддалечена *D* единици од влезот, мерено долж таванот. Од ова јаже би започнал перформансот. Задачата на циркузантот е да стигне од ова јаже до другиот крај на циркуската сала: најмалку *M* единици од влезот. При придвижувањето од едно на друго јаже, помошното јаже не се разликува од кое било друго регуларно јаже. Вашата програма треба да одговори на прашањето: кое е минималното вертикално растојание од таванот на кое што циркузантот треба да биде фатен на помошното јаже (на почетокот), за да може да ја заврши својата задача.







Да разгледаме пример со 3 перманентни јажиња, поставени на оддалеченост од 0, 3 и 6 единици од влезот, соодветно. Целта е да се достигне $M = 8$.

Нека помошното јаже е поставено на 4 единици од влезот. Замислете дека циркузантот е фатен за помошното (задебелено) јаже на вертикално растојание од 3 единици од таванот. Циркузантот може да го достигне јажето кое што е на 6 единици од влезот со нишање.

Штом ќе се фати за јажето што е оддалечено 6 единици од влезот, тој почнува да се искачува по првото јаже (во овој случај – помошното јаже).

Откако ќе се искачи по првото јаже, тој се фаќа за второто јаже на вертикално растојание од 6 - 4 = 2 единици од таванот. Сега циркузантот е спремен за нишање и точно може да ја достигне целта која е оддалечена 8 единици од влезот.

Задача

Треба да имплементирате две функции: `init`, која се повикува еднаш на почетокот од извршувањето на програмата и во која се процесираат почетните влезни параметри, и `minLength`, која дава одговор на прашањето дадено во задачата за дадена оддалеченост на помошното јаже. Функцијата `minLength` ќе се повика повеќе пати во програмата оценувач.

- `init(N, M, P[])`
  - N: бројот на јажиња
  - M: растојанието, изразено во единици, кое треба да го достигне циркузантот
  - `P[]`: низа со големина *N* од оддалеченостите на секое од јажињата од влезот на салата, мерено долж таванот на салата
- `minLength(D)`
  - D: Оддалеченоста на помошното јаже од влезот на салата, мерено долж таванот на салата

Имплементациски детали

Треба да предадете само една датотека именувана `circus.cpp`. Во оваа датотека треба да се имплементирани функциите `init` и `minLength` како што е опишано погоре. Потписите на функциите се:

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

Датотеката **circus.cpp** НЕ ТРЕБА да има функција *main(),* но може да содржи други декларации и функции кои ви се потребни во имплементацијата на функциите *init* и *minLength.* Вашата програма треба да ја содржи директивата `#include "circus.h"` на самиот почеток.

Ограничувања

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Пример

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

`minLength(4)` треба да врати 2. Циркузантот може да скокне од помошното јаже на јажето на оддалеченост 6 од влезот на салата (позиција 6), на вертикално растојание 2 единици од таванот и потоа да дојде до $M = 8$. Друга можност би била циркузантот да скокне од помошното јаже до јажето на позиција 0, па на позиција 3, па на позиција 6 и на крај да дојде до $M = 8$. Но, за реализација на оваа можност на почетокот циркузантот треба да биде фатен на помошното јаже на вертикално растојание 4 единици од таванот.

`minLength(5)` треба да врати 3. Циркузантот може да ја достигне целта веднаш од помошното јаже.

Подзадачи

Подзадача	Поени	N	minLength повици	Забелешка
1	11	$0 \le N \le 100\,000$	**1**	Само еден повик до minLength со 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 ≤ *N* ≤ 100 000**	**1 000**	
6	9	**0 ≤ *N* ≤ 100 000**	**1 000 000**	

Локално тестирање

За да можете да ги тестирате вашите функции *init* и *minLength* на вашиот локален компјутер, ќе ги добиете документите *Lgrader.cpp* and *circus.h*. Компајлирајте ги заедно со вашиот документ **circus.cpp** и ќе добиете програма која што можете да ја користите за тестирање на вашите функции.

Во Lgrader се чита влезот во следниот формат:

- линија 1: два броја *N* и *M*
- линии **2 + *i*** ($0 \le i \le N-1$): P_i
- линија $N + 2$: *Q* – колку пати ќе се повика `minLength`
- линии ***N* + 3 + *i*** ($0 \le i \le Q-1$): број *D* – параметар за секој повик на функцијата `minLength`

Во Lgrader ќе се отпечатат *Q* броеви, по еден во секоја линија, кои го претставуваат резултатот кој се добива со повик на функцијата `minLength`.

Пример за локално тестирање

Влез	Излез
3 8 0 3 6 2 4 5	2 3

СРЕЌА

Монетарниот систем во една земја е малку чуден. Постојат банкноти со можни вредности – сите цели броеви од 1 до *M*. Има уште едно чудно правило во продавниците на оваа земја – клиентот никогаш не добива кусур, но исто така не може ни да остава бакшиш – со други зборови, клиентот секогаш мора да ја плати точната вредност на неговите нарачки. Ако клиентот ја нема точната сума за неговата нарачка, тогаш тој не може да купи. Замислете само каква непогодност претставува ова за клиентите!

Климентина е девојче од горе споменатата земја. Како и сите други луѓе, таа постојано се бори против претходно опишаните правила. Климентина секогаш ја знае нејзината секвенца од банкноти – да претпоставиме дека вредностите на банкнотите се a_1, a_2,…,a_N, при што сите овие вредности се помеѓу 1 и *M*. Климентина може да поседува по повеќе од една банкнота со иста вредност. Исто така, секвенцата од вредности a_1, a_2,……,a_N, не е подредена на ниеден начин. Климентина се чувствува среќно ако, при влегување во дадена продавница, може да купи каква било комбинација од производи со вкупна цена еднаква на кој било број помеѓу 1 и збирот од вредностите на нејзините банкноти $a_1 + a_2 + ... + a_N$. На ваков начин, кога купува, таа мора да ја зема во предвид само нејзината вкупна сума на пари без да прави комплицирани пресметки за тоа дали таа може (или не може) да купува со нејзините банкноти.

Забелешка: Да ја подредиме секвенцата a_1, a_2,……,a_N, во растечки редослед. Да ја воведеме ознаката $S_i = 1 + a_1 + a_2 + ……. + a_i$. Потребен и доволен услов за секој број помеѓу 1 и $a_1 + a_2 + ……. + a_N$ да може да се претстави како збир од елементите a_1, a_2,……,a_N е да важи следново неравенство: $S_i \geq a_{i+1}$, за секое $i > 1$, при што $a_1 = 1$.

Како што и би се очекувало, секвенцата од банкноти на Климентина се менува при секоја нарачка и при секоја плата што ќе ја прими таа – затоа нејзината среќа претставува променлива. Помогнете и на Климентина со тоа што ќе напишете соодветна програма. Вашата програма на влез ќе ја прима почетната секвенца од банкноти на Климентина, како и настаните кои што се случиле – нарачки и плати. Програмата треба

да определува дали Климентина е среќна на почетокот и по случувањето на секој од настаните.

Треба да забележиме дека Климентина се чувствува среќно и кога нема никакви пари – во ваков случај таа не оди на шопинг, туку на џогирање.

Задача

Напишете ги функциите *init()* и *is_happy()*, кои ќе бидат компајлирани со оценувачот (анг. grader). Овие функции треба да служат за определување дали Климентина е среќна на почетокот и по случувањето на секој настан. Функциите ќе ги примаат како параметри почетната секвенца од банкноти и секвенците од банкноти што се отстрануваат од тековната секвенца (при нарачки) односно што се додаваат на тековната секвенца (при примање на плата).

Имплементациски детали

На системот за оценување треба да предадете еден документ со име **happiness.cpp,** кој ќе ги содржи функциите:

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

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

Опис на параметрите:

coinsCount – број на банкноти што се примени (почетна секвенца или плата) или отстранети (купување).

maxCoinSize – максимална вредност на една банкнота.

coins[] – низа, во која во случаен редослед се дадени вредности на банкноти (индексите започнуваат од 0).

event – тип на настанот:

−1 – Купување;

1 – Примање на плата.

Функцијата *init* се повикува еднаш од страна на оценувачот на почетокот – за да се иницијализира почетната секвенца од банкноти на Климентина, а потоа следуваат *Q* повици на функцијата *is_happy* со *event* = −1 (купување) или *event* = 1 (плата). По секој повик, повиканата функција треба да врати вредност *true*, ако Климентина се чувствува среќно со нејзината тековна секвенца од банкноти, или вредност *false* – во спротивниот случај.

Документот **happiness.cpp** НЕ ТРЕБА да содржи функција *main()*, но може да содржи други декларации и функции, потребни за правилно работење на функциите *init* и *is_happy*. Вашата програма треба да ја содржи директивата `#include "happiness.h"` на самиот почеток.

Ограничувања

Нека *Nc* го означува бројот на банкноти на Климентина во кој било даден момент, а *K* – бројот на банкноти употребени во која било нарачка или плата. Тогаш, имаме:

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

Се гарантира дека при секој повик на функцијата *is_happy* со *event* = −1 (купување) низата од банкноти дадена во *coins*[] ќе биде подмножество од тековната секвенца од банкноти на Климентина.

Пример

Повикана функција	настан	coinsCount	maxCoinSize	coins[]	Функцијата враќа
init		5	100	4 8 1 2 16	true
is_happy	−1 (купување)	2		2 8	false
is_happy	1 (примање на плата)	3		7 9 2	true

Подзадачи

Подзадача	Поени	Nc	M	Q
1	10	≤ 300	≤ 100	≤ 100
2	20	≤ 20000	≤ 10^{12}	≤ 1000
3	30	≤200000	≤ 1000000	≤ 100000
4	40	≤ 200000	≤ 10^{12}	≤ 100000

Локално тестирање

За да можете да ги тестирате вашите функции *init* и *is_happy* на вашиот локален компјутер, ќе ги добиете документите *lgrader.cpp* и *happiness.h*. Компајлирајте ги заедно со вашиот документ **happiness.cpp** и ќе добиете програма која што можете да ја користите за тестирање на вашите функции.

Програмата ќе го очекува следниот формат на влезни податоци:

Во првата линија се дадени два позитивни цели броеви *N* и *M* – бројот на почетни банкноти на Климентина и максималната вредност на која било банкнота.

Во втората линија се дадени *N* позитивни цели броеви, разделени меѓу себе со по едно празно место – вредности на банкнотите од почетната секвенца.

Во третата линија е даден ненегативен цел број *Q* – бројот на настани.

Во секоја од следните *Q* линии е опишан по еден настан – најпрвин, дадена е вредност за настанот: −1 (купување) или 1 (примање на плата). Во продолжение, даден е позитивен цел број *K* – број на банкноти што се отстрануваат или додаваат од/во

секвенцата на Климентина. На крајот, дадени се *K* цели броеви, разделени меѓу себе со по едно празно место – вредности на банкнотите кои што се отстрануваат или додаваат.

На стандарден излез програмата треба да отпечати *Q*+1 линија со 0 или 1 – состојбите на "среќата" на Климентина на почетокот и по случувањето на секој од настаните.

Пример за локално тестирање

Влез	Излез
5 100 4 8 1 2 16 2 -1 2 2 8 1 3 7 9 2	1 0 1

Супер ХО

ХО е игра за двајца играчи која има многу едноставни правила. Оваа игра се игра на правоаголна табла која има 3 редици и 3 колони (на почетокот таа е празна). Во првиот потег првиот играч избира едно поле од таблата и во него во него го внесува својот знак - икс (‘X’). Потоа, вториот играч избира празно (незафатено) поле и во него го запишува својот знак – нула (‘O’). Во следниот потег, првиот играч избира празно поле и запишува икс (‘X’) итн. Оваа постапка се повторува така што наизменично играат двајцата играчи. Играта ќе заврши кога некој играч ќе има внесено три свои знаци во истата редица, колона или во некоја од дијагоналите. Тој играч е победникот, а играта има победничка конфигурација. Ако таблата се исполни и нема победничка конфигурација тогаш велиме дека играчите играле нерешено.

Елена и Христијан играат модифициран верзија на играта ХО наречена супер ХО. Наместо стандардна 3x3 табла, почетната табла има бесконечно многу редици и колони. Откако првиот играч ќе го изигра првиот потег (да се забележи дека не е важно каде овој потег ќе се изигра), во следниот потег вториот играч мора да се избере поле така што заедно со полето избрано во првиот потег двете зафатени полиња се сместени во некоја 3x3 подтабла од таблата. Со други зборови, во вториот потег играчот мора да избере поле кое е најмногу две колони и/или редици оддалечено од полето избрано во првиот потег. Следните потези се прават со слична логика: играчите мора да се погрижат дека со секој нивниот потег сите зафатени полиња се сместени во некоја 3x3 подтабла од таблата. Забележете дека со секој следен потег се намалува бројот на валидни полиња каде може да се одигра следниот потег се додека да се намали таблата на стандардната 3x3 табла (под претпоставка дека никој од играчите не победил до тој момент). Погледнете го примерот каде е прикажана една игра.

Исто како во оригиналната игра ХО, победникот на супер ХО е играчот кој прв ќе зафати три соседни полиња од истиот ред, колона или дијагонала. Слично, ако сите валидни полиња се зафатени и никој не е победник, велиме дека играчите играат нерешено.

Во прилог е пример на една игра:

. . . .	. . . . 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

Во примерот првиот играч (‘X’) е победник. Да забележиме дека со подобра стратегија на вториот играч (‘O’), првиот немаше да победи.

За еден потег ќе велиме дека е *оптимален потег* ако со тој потег:

играчот ќе победи, ако тој може да победи,

играчот ќе одигра нерешено, ако тој не може да победи, а може да одигра нерешено,

или играчот ќе изгуби, во секој друг случај.

Се претпоставува дека двајцата играчи играат оптимално, т.е. со оптимални потези.

Задача

Напиши програма со која за дадена состојба на таблата ќе најдете кое поле може следниот играч да го избере ако тој игра оптимално (со оптимален потег).

Влез

Во првата линија од стандардниот влез се запишани два цели броја **N** и **M** – бројот на валидни редици и колони, соодветно. Во следните **N** линии се запишани стрингови со должина **M**, со кои се опишани редовите од таблата. Во сите примери е прикажана точната состојба на таблата каде барем еден потег е веќе изигран и играта не е завршена.

Излез

Во првата линија од стандардниот излез се наоѓаат два цели броја одделени со празно место: **R** и **C**. Тие ги претставуваат редниот број на редот и колоната, соодветно, на полето кое може да ги избере следниот играч ако тој игра оптимално (првата редица/колона има реден број 1). Ако се можни повеќе решенија, тогаш во излезот треба да прикажете било кој од нив.

Ограничувања

- $3 \leq \mathbf{N}, \mathbf{M} \leq 5$
- Сите знаци со кои се опишува таблата се од множеството {'.' (точка), 'X' (знакот големо X), 'O' (знакот големо O)}.
- Во тест случаите кои носат 50% од поените за задачата важи дека **N** = **M** = 3 (т.е. таблата е стандардната табла за ХО).

Оценување

Тестовите на оваа задача се групирани во 4 подзадачи. Секоја подзадача носи 25 поени ако сите тестови во подзадачата се точни.

Пример

Влез	Излез
3 4 .XX. .... .O..	1 1

Објаснување

На ред е вториот играч (треба да го внесе знакот ‘O’). Ако го избере полето во првата редица и четвртата колона (1,4), тогаш првиот играч може да победи ако внесе ‘X’ во полето (2, 3). По овој потег без разлика како ќе одигра вториот играч, играчот ‘X’ ќе победи. На пример ако вториот стави ‘O’ во полето (3,3) , првиот ќе стави ‘X’ во (3,4) и ќе победи.

Ако вториот играч го избере полето (1, 1), во следниот потег валидни ќе бидат полињата во колоните 1..3 и во овој случај ‘X’ нема да може да победи. Оттука, ако првиот играч го избере полето (1,1) играта може да заврши нерешено. Ова е оптимален потег на вториот играч (да забележиме дека за дадената состојба на таблата тој не може да победи).