Цирк

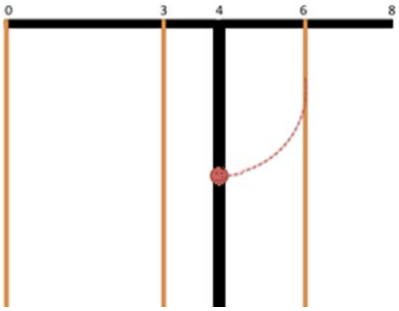
Нов цирков номер включва използването на въжета, които са закачени в права линия на тавана на залата. Входът на залата също се намира на правата, върху която са закачени въжетата. Таванът е висок 10^{18} единици, и въжетата висят свободно и стигат до пода. Целта на акробатите е да се движат от въже на въже, за да стигнат на разстояние поне M единици от входа.

Всички въжета са *N* на брой. *i*-тото въже виси на място, отдалечено на P_i единици от входа на залата, измерено по тавана.

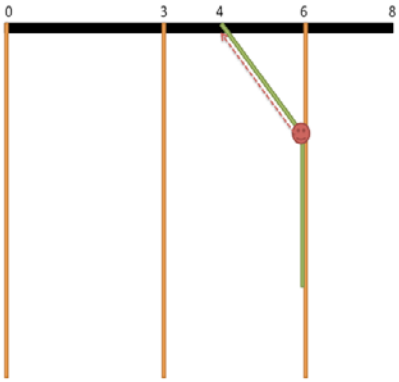
Акробатите са предпазливи и не скачат лудо от въже на въже. Придвижването става по следния начин: акробатът започва на едно въже и се държи на определено разстояние S от тавана. Акробатът може да залюлее въжето, така че да достигне други въжета, но не може да се спуска надолу по въжето, за което се държи. Акробатът може да се люлее, докато въжето не стане успоредно на пода и допре тавана. Формално, ако захватът му е на разстояние *S* от тавана, акробатът може да достигне най-много отместване *S* по посока към входа или към изхода. Ако по време на люлеенето акробатът се окаже в точка, отдалечена от входа на не по-малко от *M* единици, можем да считаме, че акробатът е изпълнил задачата си. Докато въжето (с номер *i*), за което акробатът е хванат, се люлее, той може да улови друго въже (с номер *j*), без да изпуска това, за което се държи. Тогава акробатът започва да се катери нагоре по въжето *i*, докато не достигне тавана, където може да се задържи, да си почине, но най-вече – да опъне по тавана въжето *j* до себе си, преди да продължи придвижването. Оттам нататък акробатът се залюлява на въжето, което току що е уловил (с номер *j*), заловен за точката от него, до която е стигнал при опъването.

Формално, ако акробатът виси на въже *i* на разстояние от тавана S, той може да достигне тези въжета *j*, за които $Pi - S \le Pj \le Pi + S$. При преминаване на въже *j*, радиусът на люлеенето ще бъде $|Pi - Pj|$ от тавана.

Номерът, който мениджърът на циркът иска да постави, е да се спусне акробат с допълнително въже от позиция *D* на тавана. Движението започва именно от това въже. Акробатът отново трябва да стигне на разстояние, не по-малко от *M* единици от входа. При придвижването, допълнителното въже е неразличимо от останалите въжета.

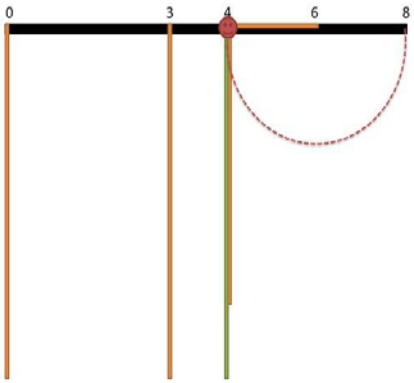


Напишете програма, която отговаря на въпроса: каква е минималната височина в единици от тавана, където трябва да бъде захванат акробатът върху допълнителното въже, така че да изпълни задачата си.



Да разгледаме пример с три постоянни въжета, съответно на позиции 0, 3 и 6 единици от входа. Целта е да се достигне *M* = 8.

Да разгледаме допълнително въже на 4 единици от входа. Нека акробатът се захване към това въже (по-плътното на фигурата) на разстояние 3 единици от тавана. Акробатът може да достигне чрез люлеене въжето, което е на 6 единици от входа.



Щом го залови, започва да се катери по допълнителното, без да изпуска хванатото. След като достигне тавана, опъва въже 6 към себе си и може да започне люлеене върху него с радиус

6 − 4 = 2 единици. При това люлеене целта (разстояние, не по-малко от 8 единици от входа) се постига и задачата е изпълнена.

Задача

Трябва да създадете две функции: `init`, която се вика веднъж в началото на програмата и обработва началните входни параметри, и `minLength`, която отговаря на едно запитване при зададена позиция на допълнителното въже. Функцията `minLength` ще бъде извиквана много пъти от програмата на журито.

- `init(N, M, P[])`
  - N: брой въжета
  - M: крайната цел – отместване от входа в единици
  - P[]: масив с размерност *N*, който съдържа позициите на всички въжета, измерени по тавана от входа
- `minLength(D)`
  - D: разстояние в единици по тавана от входа до мястото, където ще бъде спуснато допълнителното въже

Детайли по реализацията

Трябва да представите един файл с име `circus.cpp`. Той съдържа реализация на функциите `init` и `minLength`, описани по-горе, със зададените по-долу прототипи.

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

Файлът **circus.cpp** НЕ трябва да съдържа функция *main()*, но може да съдържа декларации и други функции, които са необходими за работата на *init* и *minLength.* В началото си Вашият файл трябва да съдържа `#include "circus.h"`.

Ограничения

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Пример

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

`minLength(4)` трябва да връща 2. Възможно е акробатът да прескочи от допълнителното въже към въжето на позиция 6, да се захване на две единици от тавана и да достигне целта $M = 8$. Друга възможна поредица от скокове би била от допълнителното въже към това на позиция 0, после към 3, после към 6 и така да се стигне до целта $M = 8$. Тази поредица обаче би изисквала акробатът да се захване за допълнителното въже на позиция поне 4 от тавана.

`minLength(5)` трябва да връща 3: акробатът може да достигне целта си и направо със залюляване на допълнителното въже.

Подзадачи

подзадача	точки	N	повиквания на minLength	забележка
1	11	$0 \le N \le 100\,000$	**1**	Само едно локално повикване с 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**	

Локално тестване

За да можете при желание да тествате функциите си *init* и *minLength* на локалния компютър, ви се предоставят файловете *Lgrader.cpp* и *circus.h*. Компилирайте ги заедно с Вашия файл **circus.cpp** и ще получите програма, която можете да ползвате за тестване на функциите си.

Lgrader чете от стандартния вход в следния формат:

- ред 1: две цели числа N и M
- редове $\mathbf{2 + i}$ (за $0 \le i \le N-1$): P_i
- ред $N + 2$: Q – брой пъти, в които ще се повика функцията `minLength`
- редове $\mathbf{N + 3 + i}$ ($0 \le i \le Q-1$): числата D – параметри при повикването на `minLength`

Lgrader извежда Q числа (по едно на ред), които показват връщаните стойности от повикванията на `minLength`.

Пример за локално тестване

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

ЩАСТИЕТО НА КУПУВАЧА

Колкото и странно да изглежда, в държавата Х-ландия се използват банкноти със стойности всичките цели числа от 1 до *M*. В същата държава се наблюдава още едно странно правило – в магазините винаги искат на касата да се дава сума точно такава, каквато е стойността на покупките, тъй като връщането на ресто е непосилна задача за продавачите в Х-ландия. Ако купувачът не разполага с точна сума, той не може да купи желаните стоки. Представете си какви неудобства създава това за купувачите. Всеки път, когато човек избира какви стоки да купи, той трябва усилено да съобразява дали ще може да ги плати точно на касата.

Ния живее в Х-ландия и, както всички други, изпитва неудобствата от глупавото правило за пазаруване. Ния постоянно знае с какъв набор от банкноти разполага – нека техните стойности са $a_1, a_2, \ldots\ldots, a_N$. Тези стойности са между 1 и *M*, сред тях може да има повтарящи се (банкноти с една и съща стойност) и те не са подредени по никакъв начин. Ния се чувства истински щастлива, ако знае, че наборът от банкноти, с който разполага в момента, е такъв, че влизайки в някакъв магазин, може да накупи всякакви стоки, чиято обща стойност е между 1 и общата сума, с която разполага, т.е. $a_1 + a_2 + \ldots\ldots + a_N$ – в такъв случай, избирайки стоките, тя трябва да мисли само да не надхвърли общата сума, а не дали ще може да плати избраните стоки на касата или не.

Забележка: Да сортираме $a_1, a_2, \ldots\ldots, a_N$ в ненамаляващ ред. Нека $S_i = 1 + a_1 + a_2 + \ldots\ldots + a_i$. Необходимо и достатъчно условие всяко число между 1 и $a_1 + a_2 + \ldots\ldots + a_N$ да може да се представи като сума от елементи на набора $a_1, a_2, \ldots\ldots, a_N$ е изпълнението на неравенството $S_i \geq a_{i+1}$ за всяко $i>1$ и $a_1 = 1$.

За съжаление наборът от банкноти се променя след всяко пазаруване и след всяко получаване на заплата, така че щастието на Ния е доста променлива величина. Вие можете да помогнете на момичето, като напишете програма, която, знаейки първоначалния набор от банкноти, с който е разполагала Ния, и последователността от събития, изразяващи се в пазаруване или получаване на заплата, определя нейното състояние (щастлива или не) в началото и след всяко събитие.

Трябва да се отбележи, че Ния се чувства щастлива и когато няма никакви пари – тогава твърдо знае, че не може да купи нищо.

Задача

Напишете функции *init()* и *is_happy()*, които ще се компилират заедно с програмата на журито и, получавайки информация за първоначалния набор от банкноти, с който е разполагала Ния, както и за банкнотите, които дава при всяко пазаруване или получава при всяка заплата, определят състоянието ѝ – първоначално и след всяко събитие.

Детайли по реализацията

Вие трябва да напишете и предадете към системата програмен файл **happiness.cpp**, който съдържа функциите:

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

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

Функцията *init* се извиква в началото на програмата, а *is_happy* на всяка промяна. Параметрите на функциите имат следния смисъл:

coinsCount – брой банкноти, които Ния притежава в началото или брой банкноти, с които ще пазарува или получава.

maxCoinSize - максималната стойност на една банкнота.

coins[] – масив, в който в произволен ред са записани номиналите на банкнотите, които дава или получава Ния. При *init* това са стойностите на банкнотите, с които разполага момичето в началото. Индексирането започва от 0.

event – тип на събитието: −1 – пазаруване; 1 – получаване на заплата.

Функцията *init* ще се извика само веднъж в началото на програмата. След това ще се извика Q пъти функцията *is_happy*. След всяко извикване на функция, тя трябва да

върне резултат *true* – ако Ния е щастлива с набора банкноти, с който разполага, или *false* – ако е нещастна.

Файлът **happiness.cpp** НЕ трябва да съдържа функция *main()*, но може да съдържа декларации и други функции, които са необходими за работата на *init* и *is_happy*. В началото си Вашият файл трябва да съдържа `#include "happiness.h"`.

Ограничения

Нека с *Nc* означим броя банкноти, с който разполага Ния във всеки момент, а с *K* – броя банкноти, които тя дава при пазаруване или получава като заплата.

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

Гарантирано е, че при викане на функция *is_happy* със стойност -1 за параметъра *event* (пазаруване), списъкът от банкноти, който е зададен в масива *coins*[] се съдържа в набора от банкноти, с който разполага Ния в този момент.

Пример

Функция, която се вика	event	coinsCount	maxCoinSize	coins[]	функцията връща
init		5	100	4 8 1 2 16	true
is_happy	-1(пазаруване)	2		2 8	false
is_happy	1(получаване на заплата)	3		7 9 2	true

Подзадачи

Подзадача	Точки	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 тестване

За да можете да тествате Вашите функции *init()* и *is_happy()* на локалния си компютър, Ви се предоставят файлове *Lgrader.cpp* и *happiness.h*. Компилирайте ги заедно с вашия файл **happiness.cpp** и ще получите програма, с която да тествате функцията си. Програмата изисква от стандартния вход да се въведе следната последователност от данни:

От първия ред се въвеждат две цели положителни числа *N* и *M* – брой на банкнотите, с които първоначално разполага Ния и максималната стойност, която може да има една банкнота.

Следва един ред, който съдържа *N* цели положителни числа, разделени с интервали – стойностите на банкнотите в първоначалния набор, с който разполага Ния.

От следващия ред се въвежда цяло неотрицателно число *Q* – брой на събитията, изразяващи се в пазаруване или получаване на заплата.

Следват *Q* реда, всеки от които съдържа информация за едно събитие. Първото число от реда определя вида на събитието: −1 – пазаруване, 1 – получаване на заплата. Отделено от вида на събитието с интервал, следва цяло положително число, указващо броя на банкнотите, които Ния е заплатила при пазаруване или е получила при заплата. Следва интервал и след него, разделени с интервали, следват стойностите на самите заплатени или получени банкноти.

На изход програмата ще изведе *Q* + 1 реда, всеки от които съдържа 0 или 1 – състоянията на Ния в началото и след всяко събитие.

Пример за локално тестване

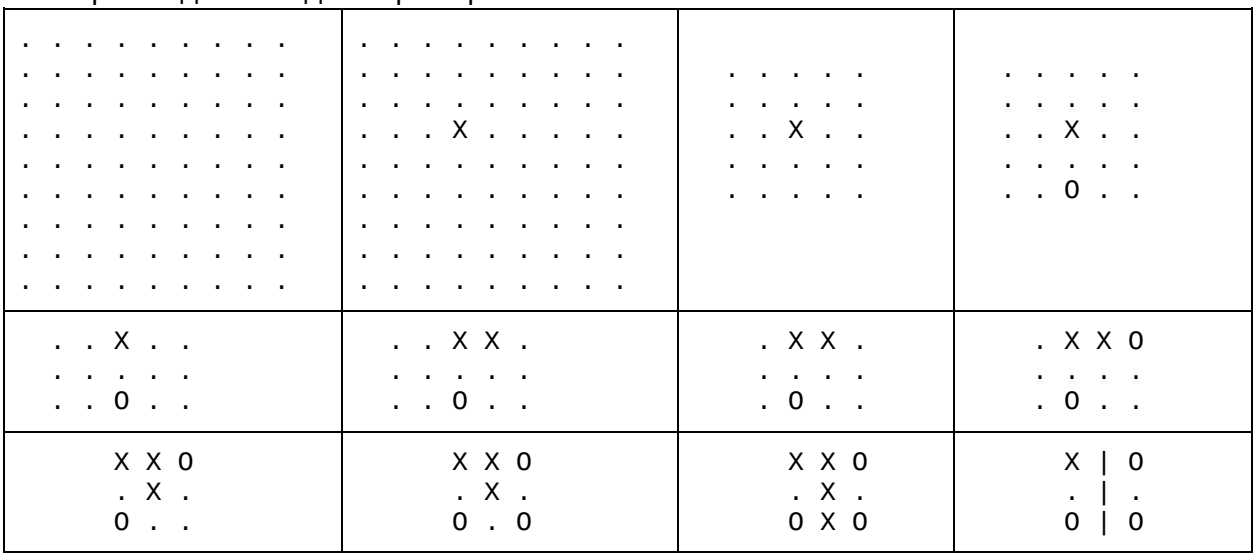
Вход	Изход
5 100 4 8 1 2 16 2 -1 2 2 8 1 3 7 9 2	1 0 1

Нов морски шах

Морски шах е игра за двама с изключително прости правила. В началото се започва на празна дъска с три реда и три колони. Първият играч избира една от клетките и я запълва със своя знак ('X'). След това вторият играч избира празна клетка и я запълва със своя знак ('O'). На третия ход отново първият играч ('X') избира празна клетка и т.н. Играчите се редуват и не могат да пропускат ход (а и няма смисъл да го правят). Играта продължава, докато някой от играчите сложи три свои знака на един и същи ред, една и съща колона или някой от двата диагонала. Ако дъската се запълни и никой от играчите не е победил, играта завършва наравно.

Ели и Крис играят на модифицирана версия на играта. Вместо да се играе на стандартната 3x3 дъска, в началото се започва с безкрайно голяма дъска. След като първият играч направи своя ход (реално без значение къде), следващият ход може да бъде само на такова поле, което потенциално би могло да бъде на една и съща (3x3) дъска с първия. С други думи, вторият ход трябва да е най-много на разстояние 2 по хоризонтала и вертикала от първия ход. Следващите ходове следват сходна логика - играчите винаги трябва да избират полета по такъв начин, че всички досега избрани клетки да могат да бъдат събрани в стандартна дъска 3x3. Както и при обикновения морски шах, играчът, който първи задраска три съседни полета по хоризонтал, вертикал или диагонал, печели. Ако никой от играчите не успее да направи това преди възможните полета да се изчерпят, играта завършва наравно. Забележете, че колкото повече напредва играта, толкова повече се смалява дъска (полетата, на които може да се играе), докато накрая не стигне до стандартните размери 3x3 (стига никой от играчите да не спечели преди това).

Нека разгледаме следния пример:



В примера първият играч ('X') печели, но при оптимална игра на втория играч ('O') това не би било така.

Оптимална игра ще наричаме правенето на такива ходове, с които играчът на ход печели, ако може да се спечели или постига "равен", ако не може да се спечели, но може да се направи равен. Ако не може да се спечели или да се постигне равенство, всеки ход се смята за оптимален. Приемаме, че противникът също играе оптимално.

Задача

Напишете програма, която по текущо състояние на дъската намира коя клетка трябва да избере играчът, който е на ход, за да играе оптимално.

Вход

Първият ред на стандартния вход съдържа две цели числа **N** и **M,** задаващи съответно броя оставащи валидни редове и колони от дъската. Следват **N** реда, всеки с по **M** символа, всеки от които описва следващия ред от дъската. Тези данни представляват коректно описание на текущото състояние на дъската. Гарантирано е, че ще е направен поне един ход и играта няма да е завършила.

Изход

Програмата извежда на единствен ред от стандартния изход две цели числа **R** и **C** – съответно ред и колона (индексирани от едно). Това е позицията на клетка, която трябва да избере играчът, който е на ход, при оптимална игра. Ако има повече от една възможност, изходът може да описва която и да е от тях.

Ограничения

- $3 \leq \mathbf{N}, \mathbf{M} \leq 5$
- Всички символи, описващи дъската, са от азбуката {'.', 'X', 'O'} (главни латински букви 'X', 'O').
- В тестове, даващи 50% от точките за задачата, **N** = **M** = 3 (тоест ще е стандартен морски шах).

Оценяване

Тестовете са групирани в 4 подгрупи. Всяка подгрупа получава 25 точки, само ако всичките тестове в нея са решени правилно.

Пример

Вход

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

Изход

```
1 1
```

Пояснение

Забележете, че ако играчът, който е на ход ('O'), играе в (1, 4), то 'X' би спечелил (играейки в (2, 3) и поставяйки 'O' във „вилица").

Играейки в (1, 1) вече дъската е ограничена само в колоните 1..3, като така 'X' не може да сложи в (1, 4) и да спечели. Оттам нататък има стратегия, с която 'O' може да постигне равенство.