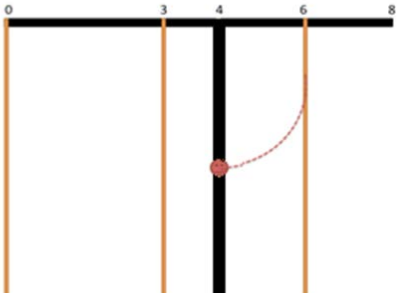
SİRK

Yeni bir sirk gösterisi, çok uzun iplerle sirkin düz tavanından sallanmayı içermektedir. Sirkin girişi ve bütün ipler düz bir çizgi üzerinde yerleştirilmişlerdir. Sirkin tavanı 10^{18} birim yüksekliğindedir, bütün ipler serbestçe sallanmaktadır ve her bir ip sirkin tabanına kadar uzanmaktadır. Cambaz bir ipten diğerine geçerek sirk girişinden en az M birim uzaklaşmayı hedeflemektedir.

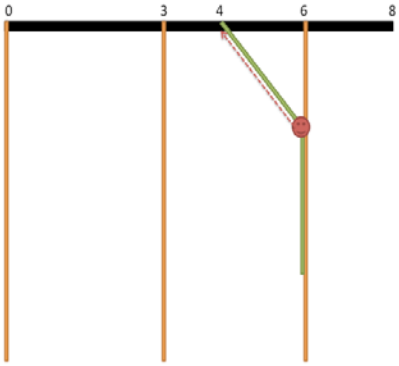
Toplam N tane ip vardır. i-nci ip sirk girişinden P_i birim uzaklıkta (tavan boyunca ölçülen mesafe) sirk tavanından sallanmaktadır.

Cambaz dikkatli bir kişidir ve bir ipten diğerine çılgınca atlayarak rastgele ilerlemez. Cambazın i-nci ipi tavandan S birim uzaklıkta tuttuğunu hayal edelim. Cambaz tutmuş olduğu ipte dilediği gibi sağa sola sallanabilir. İpte sallanırken, eğer cambaz girişten en az M birim uzaklaşabilirse (tavan boyunca), cambaz görevini tamamlamış olacaktır. Sallanma sırasında cambaz ilk ipten en fazla S birim uzaklıkta asılı duran başka bir ipi tutabilir. Matematiksel olarak ifade etmek gerekirse, eğer $|P_i - P_j| \leq S$ ise cambaz j-nci ipi tutabilir. j-nci ipi tuttuğu anda cambaz hem i-nci hem de j-nci ipi birlikte tutmaya devam eder. Daha sonra, j-nci ipi hala tutmaya devam ederek, i-nci ipte tırmanmaya başlar. Cambaz i-nci ipin tavana asılı olduğu yere ulaştığında, j-nci ip tavan boyunca gergin olacak şekilde kendisine çeker. Bu işlemlerden sonra cambaz hareketine sadece j-nci ipi tutarak devam eder. j-nci ip üzerinde tavandan ne kadar uzak olduğu bir önceki adımda ipin gerilmesi ile ilişkilidir. Matematiksel olarak söylemek gerekirse, cambaz j-nci ip üzerinde tavandan $|P_i - P_j|$ birim aşağıda olarak hareketine devam eder.

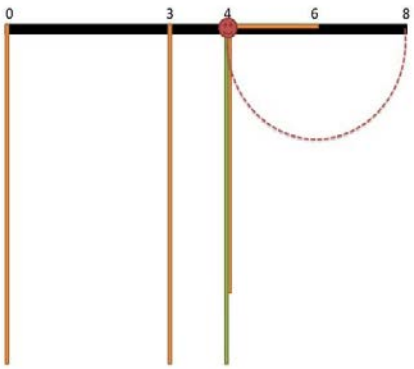
Sirk sahibi, gösteriye, girişten *D* birim uzaklıkta (tavan boyunca) ek bir geçici ip eklemek ister. Bu ip cambazın gösterisine başlayacağı iptir. Cambazın görevi, bu ipten başlayarak ve ipten ipe geçerek girişten en az *M* birim uzaklaşmaktır. İpten ipe geçiş sırasında geçici olarak gösteriye eklenen ip, diğer sabit iplerden farklı değildir. Yazacağınız program şu soruyu cevaplamalıdır: cambazın görevini tamamlayabilmesi için, gösteri başında geçici ipi tutacağı nokta tavandan en az kaç birim aşağıda olmalıdır.



3 tane sabit ip içeren bir örnek düşünelim. Bu sabit ipler, sirk girişinden 0, 3 ve 6 birim uzaklıkta tavana asılı olsunlar. Cambazın hedefi de girişten *M* = 8 birim uzaklığa erişmek olsun.



Girişten 4 birim uzaklıkta bir geçici ip düşünelim. Cambazın da bu geçici ipi (şekildeki kalın ip) tavandan 3 birim aşağıda tuttuğunu hayal edelim (minimum mesafe için Örnek kısmına bakınız). Cambaz sallanarak girişten 6 birim uzaklıkta olan ipe ulaşabilir.



Girişten 6 birim uzaklıktaki ipi tutar tutmaz cambaz birinci ipe (yani geçici ipe) tırmanmaya başlar.

Cambaz birinci ipe tırmanmayı tamamladığında, ikinci ipi tavana asılı olduğu yerden 6 - 4 = 2 birim aşağıda tutmaktadır. Daha sonra sallanmaya devam ederek, girişten 8 birim uzaklıktaki hedefine ucu ucuna ulaşabilir.

Görev

`init` ve `minLength` adında iki fonksiyon gerçekleştirmelisiniz: `init` fonksiyonu program çalışmasının başında bir kez çağrılır ve başlangıç girdi parametrelerini işlemek için kullanılır. `minLength` fonksiyonu verilen bir geçici ip için istenilen minimum mesafeyi hesaplar. `minLength` fonksiyonu grader programı tarafından birçok kez çağrılacaktır.

- `init(N, M, P[])`
  - N: iplerin sayısı
  - M: hedef mesafenin uzaklığı
  - P[]: Bütün sabit iplerin, girişten uzaklıklarını gösteren (tavan boyunca) *N* boyunda bir dizi
- `minLength(D)`
  - D: Geçici ipin girişten tavan boyunca olan mesafesi

Gerçekleştirim detayları

`circus.cpp` adında tek bir dosya göndermelisiniz. Bu dosya yukarıda açıklandığı şekilde `init` ve `minLength` fonksiyonlarını aşağıdaki protiplere göre gerçekleştirmelidir.

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

circus.cpp dosyası *main()* fonksiyonunu içermeMElidir, fakat *init* and *minLength* fonksiyonlarınız için gerekli olabilecek başka tanımlar ya da fonksiyonlar içerebilir. Programınız en tepede `#include "circus.h"` satırını içermelidir.

Kısıtlar

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Örnek

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

`minLength(4)` cevap olarak 2 döndürmelidir. Geçici ipten, girişten 6 uzaklıkta bulunan ipe atlamak için geçici ipi tavandan 2 birim aşağıda tutmamız yeterlidir. Sonra da 6 birim uzaklıktaki ipten $M = 8$'deki hedefe ulaşabiliriz. $M = 8$'deki hedefe ulaşmak için başka bir yol ise geçici ipten, önce 0 uzaklıktaki ipe, daha sonra 3 uzaklıktaki ipe ve oradan da 6 uzaklıktaki ipe atlayıp sonra hedefe ulaşmak olabilir. Ama bu ikinci yol cambazın başlangıçta geçici ipi tavandan 4 birim aşağıda tutmasını gerektirdiğinden aranan çözüm değildir.

`minLength(5)` 3 döndürmelidir. Cambazın hedefe direkt olarak geçici ipten ulaşmasına izin vardır.

Altgörevler

Altgörev	Puan	N	minLength çağırma sayısı	not
1	11	$0 \le N \le 100\,000$	1	D = 0 olan tek bir minLength sorgusu
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	

Kendi bilgisayarınızda test etme

init ve *minLength* fonksiyonlarınızı kendi bilgisayarınızda test etmek için *Lgrader.cpp* ve *circus.h* dosyalarını kullanmanız gerekmektedir. **circus.cpp** dosyanız ile birlikte bu dosyaları derleyerek fonksiyonlarınızı test etmek için kullanabileceğiniz bir program elde edebilirsiniz.

Lgrader girdiyi aşağıdaki formatta okur:

- satır 1: *N* ve *M* tamsayıları
- **2 + *i*** ($0 \le i \le N-1$). satırlar: P_i
- satır *N* + 2: *Q* - `minLength` fonksiyonunun kaç kez çağrılacağı
- ***N* + 3 + *i*** ($0 \le i \le Q-1$). satırlar: *D* sayıları – her bir *`minLength`* sorgusunun parametresi

Lgrader her bir satırda bir tane olacak şekilde *`minLength`* sorgularının sonucu olan *Q* tane tamsayı basacaktır

Kendi bilgisayarınızda alabileceğiniz bir örnek test sonucu

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

MUTLULUK

X-land ülkesinde para ve alışveriş sistemi oldukça tuhaftır. Ülkede 1 ile *M* arasında tüm tamsayı değerlerinde kağıt para vardır. Asıl tuhaf olan kural ise müşteriler dükkanlarda alışveriş yaparken para üstü alamaz ve bahşiş olarak para bırakamaz, yani müşteri aldığı malın tam olarak fiyatı ne ise o kadar para ödemesi yapmak zorundadır. Müşteri eğer satın almak istediği malın fiyatına eşit miktarda ödeme yapamıyorsa o malı alamaz. Bu kuralın müşteriler için ne kadar rahatsızlık verici olduğunu varın siz düşünün!

Niya, X-land'li bir vatandaştır. Tüm herkes gibi O da bu kurallara karşı sürekli mücadele halindedir. Niya her zaman cüzdanında hangi paraların olduğunu bilmektedir– varsayalım ki cüzdanda *N* tane kağıt para vardır ve bunların değerleri $a_1, a_2,\ldots,a_N$ dir. Her bir paranın değeri 1 ile *M* arasındadır ve cüzdanında aynı değerde birden fazla para olabilir. $a_1, a_2,\ldots\ldots,a_N$ değer bakımından sıralı olarak verilmek zorunda değildir. Niya alışveriş için dükkana girdiğinde elindeki paralardan 1 ile $a_1 + a_2 + \ldots + a_N$ arasındaki (sınırlar dahil) tüm tamsayılar için kombinasyon üretebiliyorsa mutlu aksi halde mutsuzdur. Bu durumda fiyatı 1 den elindeki toplam paraya kadar olan herhangi bir malı kurallar dahilinde alabilecektir.

Önemli Bilgi: $a_1, a_2,\ldots\ldots,a_N$ cüzdandaki paraların artan sırada sıralanmış hali olsun ve $S_i = 1+a_1+ a_2+\ldots\ldots+a_i$ olarak tanımlansın. 1 ile $a_1+ a_2+\ldots\ldots+a_N$ arasındaki tüm tamsayıları elde etmenin gerekli ve yeter şartı $S_i \geq a_{i+1}$ eşitsizliğinin *tüm $i>1$ ve $a_1 = 1$ için sağlanmış olmasıdır.*

Tahmin edileceği üzere, Niya'nın cüzdanındaki paralar her bir alışveriş ve kendisine her bir maaş ödemesinden sonra değişmektedir- böylelikle kendisinin mutluluğu da değişken olmaktadır. Kendisine bir program yazarak yardımcı olabilirsiniz. Programınız Niya'nın cüzdanındaki ilk para durumunu ve alışveriş/maaş ödemesi durumlarını girdi olarak alacaktır. Program her bir alışveriş/maaş ödemesi sonrası Niya'nın mutlu olup olmadığını belirlemelidir.

Niya'nın cüzdanında hiç para yoksa mutlu olduğu varsayılmaktadır.

Görev

Jüri'nin grader'ı ile beraber derlenecek olan *init()* ve *is_happy()* fonksiyonlarını yazınız. Bu fonksiyonlar işlem sonunda Niya'nın mutlu olup olmadığını belirleyecektir. İlk fonksiyon başlangıçta cüzdandaki paraların ne olduğunu alacak, ikinci fonksiyon ise alışveriş (cüzdandan para çıkması) ve maaş ödemesi (cüzdana para girmesi) durumlarında çıkan-giren para değerlerini girdi olarak alacaktır.

Gerçekleştirim detayları

happiness.cpp adında ve aşağıdaki fonksiyonları içeren bir dosya göndermelisiniz:

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

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

Parametre tanımları:

coinsCount –kağıt para sayısı.

maxCoinSize – Her bir kağıt paranın alabileceği en büyük değer.

coins[] – indisi 0 dan başlayan dizi, dizideki kağıt para değerleri (rasgele sırada verilir, sıralı verilmek zoruda değildir)

event – alışveriş/maaş ödemesi seçimi :

−1 değeri Alışveriş;
1 değeri Maaş ödemesini belirtir.

init fonksiyonu grader tarafından en başta bir kez çağrılır ve amacı Niya'nın cüzdanındaki kağıt paraların ilk durumunu belirlemektir. Grader daha sonra *is_happy* fonksiyonunu *Q* kez çağırır: her bir çağırma için *event* = −1 (alışveriş) or *event* = 1 (maaş ödemesi) olabilir. Fonksiyon, her bir çağırma sonrasında, eğer Niya cüzdanındaki para ile mutluysa *true*, eğer mutsuzsa *false* döndürmelidir.

happiness.cpp dosyası *main()* fonksiyonunu içermeMElidir. Fakat programınızın doğru çalışması için başka tanımlar ve fonksiyonlar içerebilir. Programınız `#include "happiness.h"` içermelidir.

Kısıtlar

Nc Niya'nın herhangi bir andaki cüzdanında bulunan kağıt para sayısını, *K* ise her bir *event* (alışveriş/maaş ödemesi) için cüzdana giren/çıkan kağıt para sayısı olsun.

$0 \le Nc \le 200\,000$
$0 \le Q \le 100\,000$
$1 \le M \le 10^{12}$
$1 \le K \le 5$
is_happy fonksiyonu *event* = −1 (alışveriş) ile çağrıldığında *coins*[] dizisinin Niya'nın cüzdanında o an yer alan kağıt para kümesinin bir altkümesi olacağı garanti edilmektedir.

Örnek

çağrılan fonksiyon	Event	coinsCount	maxCoinSize	coins[]	dönen değer
İnit		5	100	4 8 1 2 16	true
is_happy	−1(alışveriş)	2		2 8	false
is_happy	1(maaş ödemesi)	3		7 9 2	true

Altgörevler

Altgörev	Puan	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

Kendi bilgisayarınızda test etme

init ve *is_happy* fonksiyonlarını kendi bilgisayarınızda test etmek için size *lgrader.cpp* ve *happiness.h* dosyaları verilecektir. Bu dosyaları **happiness.cpp** dosyanız ile beraber derleyerek kendi testlerinizi yapabileceğiniz programı elde edebilirsiniz.

Bu programın girdi formatı şu şekildedir:

İlk satırda *N* ve *M* sayıları- Niya'nın ilk kağıt para sayısı ve bir kağıt paranın maksimum değeri.

İkinci satırda *N* tane pozitif tamsayı (arada birer boşluk)- ilk durumdaki kağıt paraların değerleri.

Üçüncü satırda *Q* değeri.

Takip eden *Q* satırın her birinde bir event (alışveriş/maaş ödemesi)- ilk olarak event tipi (-1 ya da 1), daha sonra pozitif tamsayı *K* (cüzdandan çıkan yada eklenen kağıt para sayısı), daha sonra *K* adet tamsayı (kağıt paraların değerleri).

Standart çıktıya *Q*+1 tane satır yazdırmalısınız. Her bir satırda Niya'nın mutluluk durumunu belirten 0 (mutsuz) yada 1 (mutlu) yazdırmalısınız.

Kendi bilgisayarınızda test için örnek

Girdi	Çıktı
5 100 4 8 1 2 16 2 -1 2 2 8 1 3 7 9 2	1 0 1

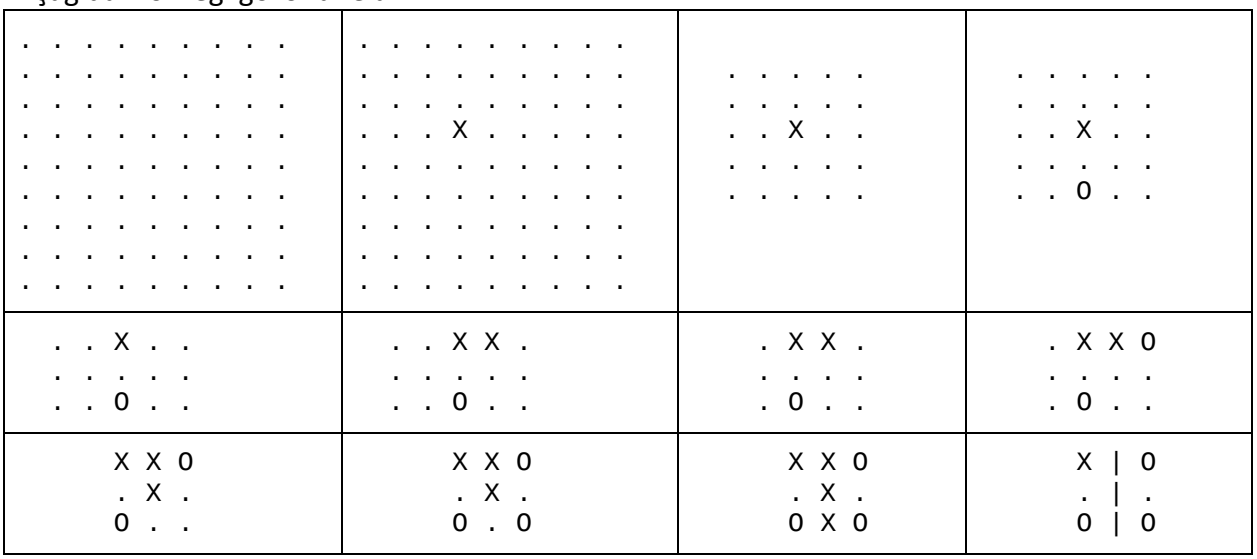
Müthiş TTT

Tic-Tac-Toe iki oyuncu arasında basit kurallar ile oynanan bir oyundur. Oyun, başlangıçta boş olan 3 satır ve 3 sütunlu bir tahta üzerinde oynanır. Oyun, birinci oyuncunun tahtanın hücrelerinden birini seçmesi ve buraya ('X') işareti koyması ile başlar. Daha sonra ikinci oyuncu boş bir hücre seçer ve buraya ('O') işareti koyar. Üçüncü turda, ilk oyuncu tekrar boş bir hücre seçer ve buraya ('X') işareti koyar vs. Oyuncular bu şekilde karşılıklı hamle yaparlar ve herhangi bir oyuncu kendi sırasını es geçemez (zaten bunu yapması da mantıklı değildir). Oyun, herhangi bir oyuncunun kendi işaretini ardışık olarak aynı satıra, aynı sütuna veya iki köşegenlerden birine koyması ile sona erer. Eğer bunu herhangi bir oyuncu başaramadan tahta dolarsa, oyun berabere biter.

Elly and Kris bu oyunun değişik bir versiyonunu oynamaktadır. 3x3'lük bir tahta yerine, oyuna başlangıçta sonsuz büyüklükteki bir tahta ile başlanır. İlk oyuncu hamlesini yaptıktan sonra (bu hamlenin nerede olduğu önemli değildir), sonraki hamle sadece bu hamlenin 3x3'lük tahtasına sığacak bir şekilde yapılabilir. Başka bir deyişle, ikinci hamle ilk hamlenin en fazla 2 sütun ve/veya satır uzaklığında olabilir. Sonraki hamle benzer bir mantık ile yapılır: oyuncular yeni hamlelerini, daha önce yapılmış tüm hamleler ile 3x3'lük tahtaya sığacak şekilde yapmak zorundadır. Not: Oyun devam ettikçe, legal hamle hücreleri daralmakta ve sonuçta standart 3x3'lük tahta oluşmaktadır (daha önce herhangi bir oyuncunun kazanmadığı varsayıldığında). Lütfen açıklama için aşağıdaki örneğe bakınız.

Orjinal oyunda olduğu gibi, kazanan oyuncu aynı satır, sütun veya köşegene üç ardışık işaretini ilk olarak yerleştirendir. Benzer bir şekilde, eğer herhangi bir oyuncu oyunu kazanmadan tüm legal hamle hücreleri dolarsa, oyun berabere biter.

Aşağıdaki örneği gözönüne alın:



Bu örnekte ilk oyuncu ('X') kazanmaktadır, fakat ikinci oyuncunun ('O') daha iyi bir strateji kullanması ile bu durum değişmektedir.

Optimal oyun şu şekilde tanımlanır: Eğer oyun kazanılabilirse, oyuncu oyunu kazanmak için gerekli hamleyi yapar. Eğer oyun kazanılamaz ancak berabere bitebilirse, oyuncu bunun için gerekli hamleyi yapar. Son olarak eğer ilk iki opsiyon geçerli değilse oyuncu oyunu kaybeder. Burada rakip oyuncunun da optimal olarak oynadığını varsayıyoruz.

Görev

Verilen bir tahtadaki hamleleri gözönüne alarak, bir oyuncunun optimal oynaması durumunda hangi hücreyi seçmesi gerektiğini bulan bir program yazınız.

Girdi

Standart girdinin ilk satırı iki tamsayı N ve M içermektedir – sırası ile geçerli satır ve sütun sayısı. Takip eden N satırın herbiri, M uzunluğunda olup tahtanın sıradaki satırının durumunu gösteren dizgi (string) içerir. Satırların tamamına bakıldığında tahtanın durumu görülebilir. En az bir tane hamlenin yapıldığı (dolayısıyla kalan geçerli hücrelerin sonlu sayıda olduğu) ve oyunun henüz bitmediği garanti edilmektedir.

Çıktı

Program standart çıktıya aralarında boşluk olan iki tane tamsayı **R** ve **C** yazar: Sırası ile satır ve sütun numarası (numaralar 1'den itibaren başlar). Bu sayılar sonraki oyuncunun optimal oynaması durumunda hamlesinin pozisyonunu belirtmektedir. Eğer birden fazla seçenek varsa, çıktı bunlardan herhangi birini verebilir.

Kısıtlar

- $3 \leq \mathbf{N}, \mathbf{M} \leq 5$
- Tahtayı tanımlayan tüm semboller şu alfabedendir {'.' (nokta), 'X' (büyük harf), 'O' (büyük harf)}.
- Testlerde, görevin %50 puanı **N** = **M** = 3 olması haline verilir (yani tahta standart Tic-Tac-Toe tahtasıdır).

Notlandırma

Problem testleri 4 tane alt göreve gruplandırılmıştır. Her bir alt görev içindeki tüm testler geçilirse, bu alt görev için 25 puan verilir.

Örnek

Girdi

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

Çıktı

```
1 1
```

Açıklama

İkinci oyuncunun (‘O’) hamle sırasıdır. Eğer bu oyuncu (1,4) seçerse, ‘X’ (2,3) pozisyonuna oynayıp ‘O’ yu çıkmaz bir pozisyona koyarak oyunu kazanabilir – ‘O’ hangi hamleyi yaparsa yapsın ‘X’ kazanan son hamleye sahip olur.

(1,1) oynanması durumunda ise sonraki hamle 1..3 sütunlarına sınırlandırılmış olur. Dolayısıyla, ‘X’ için bir hamle sonrasında (1,4)’e işaret koyarak oyunu kazanma imkansız olur. Bu şekilde, ‘O’ için oyunu berabere görürecek bir strateji oluşmuş olur.