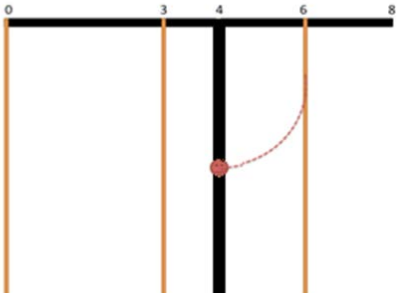
CIRCUS

Μία καινούρια παράσταση τσίρκου περιλαμβάνει τη χρήση πολύ μεγάλων σχοινιών, τα οποία κρέμονται από το ταβάνι του τσίρκου. Η είσοδος του τσίρκου και όλα τα σχοινιά είναι τοποθετημένα σε ευθεία γραμμή. Το ταβάνι είναι 10^{18} μονάδες ψηλό και τα σχοινιά κρέμονται ελεύθερα και αγγίζουν το έδαφος. Οι ακροβάτες πρέπει να κινούνται από το ένα σχοινί στο άλλο, έτσι ώστε να μπορούν να πάνε τουλάχιστον *M* μονάδες μακριά από την είσοδο.

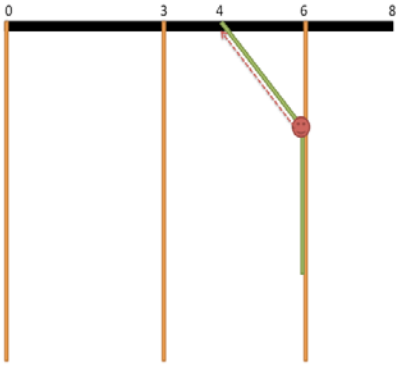
Συνολικά υπάρχουν *N* σχοινιά. Το *i-οστό σχοινί κρέμεται από μία θέση που βρίσκεται* P_i μονάδες από την είσοδο, όπως αυτό μετρήθηκε κατά μήκος του ταβανιού.

Οι ακροβάτες είναι προσεκτικοί και δεν πηδάνε απερίσκεπτα από το ένα σχοινί στο άλλο. Φανταστείτε έναν ακροβάτη να κρατάει το *i*-οστό σχοινί σε απόσταση *S* μονάδων από το ταβάνι. Ο ακροβάτης μπορεί να κάνει κούνια στο σχοινί που κρατάει. Αν κάνοντας κούνια σε κάποιο σχοινί φτάσει σε ένα σημείο όχι λιγότερο από *M* μονάδες μακριά από την είσοδο, μπορούμε να θεωρήσουμε ότι το έργο του έχει ολοκληρωθεί. Όσο κάνει κούνια, ο ακροβάτης μπορεί να πιάσει ένα άλλο σχοινί, το οποίο κρέμεται σε μία θέση η οποία είναι το πολύ *S* μονάδες μακριά από το πρώτο σχοινί. Τυπικά, μπορεί να πιάσει το *j*-οστό σχοινί αν $|P_i - P_j| \leq S$. Τώρα ο ακροβάτης κρατάει και τα δύο σχοινιά, το *i*-οστό και το *j*-οστό. Αυτή τη στιγμή, θα μπορούσε να ξεκινήσει να σκαρφαλώνει προς τα πάνω το *i*-οστό σχοινί, ενώ την ίδια ώρα κρατάει και το *j*-οστό σχοινί. Όταν ο ακροβάτης φτάσει στο σημείο, στο οποίο το *i*-οστό σχοινί κρέμεται από το ταβάνι, εξασφαλίζει ότι το *j*-οστό σχοινί είναι τεντωμένο κατά μήκος του ταβανιού. Μόνο σε αυτό το σημείο ο ακροβάτης μπορεί να συνεχίσει να κινείται κρατώντας το *j*-οστό σχοινί, σε απόσταση από το ταβάνι βασισμένη στο πώς το σχοινί τεντώθηκε στο τάβανι. Τυπικά ο ακροβάτης συνεχίζει την κίνησή του κρατώντας το *j*-οστό σχοινί σε απόσταση $|P_i - P_j|$ από το ταβάνι.

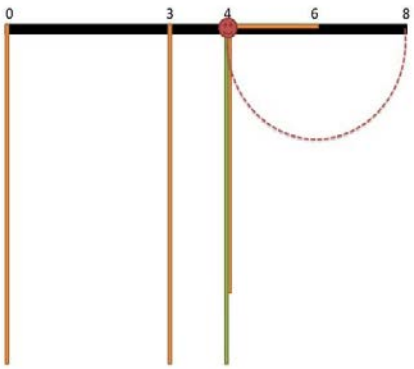
Ο διευθυντής του τσίρκου θέλει να προσθέσει ένα επιπλέον, προσωρινό σχοινί, το οποίο θα κρέμεται σε μία θέση η οποία είναι *D* μονάδες μακριά από την είσοδο, όπως αυτή έχει μετρηθεί κατά μήκος του ταβανιού. Αυτό είναι το σχοινί στο οποίο θα ξεκινήσει η παράσταση. Το έργο του ακροβάτη είναι να φτάσει από αυτό το σχοινί στο άκρο του τσίρκου: τουλάχιστον *M* μονάδες μακριά από την είσοδο. Κατά τη διάρκεια της κίνησης από το ένα σχοινί στο άλλο, το προσωρινό σχοινί δεν διαφοροποιείται από ένα κανονικό σχοινί. Το πρόγραμμά σας θα πρέπει να απαντάει την ερώτηση: ποια είναι η ελάχιστη απόσταση από το ταβάνι, στην οποία ο ακροβάτης πρέπει να κρατάει το προσωρινό σχοινί, έτσι ώστε να ολοκληρώσει το έργο του.



Θεωρήστε ένα παράδειγμα με 3 μόνιμα σχοινιά, τοποθετημένα αντίστοιχα 0, 3 και 6 μονάδες μακριά από την είσοδο. Ο στόχος είναι να φτάσει στο *M* = 8.



Θεωρήστε ένα προσωρινό σχοινί, 4 μονάδες μακριά από την είσοδο. Φανταστείτε τον ακροβάτη να κρατάει το προσωρινό σχοινί (έντονη γραμμή στο σχήμα) 3 μονάδες μακριά από το ταβάνι. Ο ακροβάτης μπορεί να φτάσει το σχοινί που βρίσκεται 6 μονάδες από την είσοδο κάνοντας κούνια.



Μόλις ο ακροβάτης πιάσει το σχοινί που είναι 6 μονάδες μακριά από την είσοδο, ξεκινάει να σκαρφαλώνει προς τα πάνω το πρώτο σχοινί (σε αυτή την περίπτωση – το προσωρινό).

Αφού σκαρφαλώσει προς τα πάνω το πρώτο σχοινί, κρατάει το δεύτερο σχοινί 6 - 4 = 2 μονάδες από τη βάση του. Τώρα ο ακροβάτης είναι έτοιμος να κάνει κούνια και μπορεί να φτάσει ακριβώς το στόχο, ο οποίος είναι 8 μονάδες μακριά από την είσοδο.

Πρόβλημα

Πρέπει να υλοποιήσετε δύο συναρτήσεις: την `init`, η οποία καλείται μία φορά στην αρχή της εκτέλεσης του προγράμματός σας και επεξεργάζεται τις αρχικές παραμέτρους εισόδου, και την `minLength`, η οποία απαντά ένα ερώτημα δεδομένης της θέσης ενός προσωρινού σχοινιού. Η συνάρτησή σας `minLength` θα κληθεί πολλές φορές από το πρόγραμμα του grader.

- `init(N, M, P[])`
  - `N`: το πλήθος των σχοινιών
  - `M`: η απόσταση στόχος από την είσοδο σε μονάδες
  - `P[]`: ένας πίνακας μεγέθους *N* ο οποίος περιέχει τις θέσεις όλων των σχοινιών, όπως αυτές έχουν μετρηθεί σε μονάδες από την είσοδο του τσίρκου κατά μήκος του ταβανιού.
- `minLength(D)`
  - `D`: η απόσταση σε μονάδες κατά μήκος του ταβανιού από την είσοδο του τσίρκου, όπου θα βρίσκεται το προσωρινό σχοινί

Λεπτομέρειες υλοποίησης

Θα πρέπει να υποβάλετε ακριβώς ένα αρχείο, το οποίο θα ονομάζεται `circus.cpp`. Αυτό το αρχείο υλοποιεί τις συναρτήσεις `init` και `minLength`, όπως αυτές περιγράφηκαν παραπάνω, ακολουθώντας τους παρακάτω ορισμούς.

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

Το αρχείο **circus.cpp** ΔΕΝ πρέπει να περιέχει συνάρτηση *main()*, αλλά μπορεί να περιέχει άλλες δηλώσεις και συναρτήσεις, απαραίτητες για τη σωστή λειτουργία των συναρτήσεων *init* και *minLength.* Το πρόγραμμά σας πρέπει στην αρχή να περιέχει το `#include "circus.h"`

Περιορισμοί

$1 \le M \le 10^9$

$0 \le P_i < M$

$0 \le D < M$

Παράδειγμα

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

Η `minLength(4)` πρέπει να επιστρέψει 2. Είναι πιθανό να μεταβεί από το προσωρινό σχοινί στο σχοινί που βρίσκεται στη θέση 6, κρατώντας το 2 μονάδες από το ταβάνι, και στη συνέχεια να φτάσει το $M = 8$. Μία άλλη πιθανή ακολουθία από μεταβάσεις θα ήταν να μεταβεί από το προσωρινό σχοινί στο σχοινί που βρίσκεται στη θέση 0, μετά στο 3, μετά στο 6 και μετά να φτάσει το $M = 8$. Παρ' όλα αυτά, η δεύτερη ακολουθία από μεταβάσεις θα απαιτούσε από τον ακροβάτη στο ξεκίνημα να κρατά το προσωρινό σχοινί 4 μονάδες από το ταβάνι.

Η `minLength(5)` πρέπει να επιστρέψει 3. Ο ακροβάτης επιτρέπεται να φτάσει την έξοδο ακόμη και κατευθείαν από το προσωρινό σχοινί.

Υποπροβλήματα

Υποπρό-βλημα	Μονάδες	*N*	Κλήσεις της minLength	Σημείωση
1	11	$0 \le N \le 100\,000$	**1**	Μόνο μία κλήση της minLength με 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**	

Τοπικός έλεγχος

Για να μπορέσετε να ελέγξετε τις συναρτήσεις σας *init* και *minLength* στον τοπικό σας υπολογιστή σας δίνονται τα αρχεία *Lgrader.cpp* και *circus.h*. Μεταγλωττίστε τα μαζί με το αρχείο σας **circus.cpp** και θα λάβετε ένα πρόγραμμα το οποίο μπορείτε να χρησιμοποιήσετε για τον έλεγχο των συναρτήσεών σας.

Το *Lgrader* διαβάζει την είσοδο με την παρακάτω μορφή:

- γραμμή 1: δύο ακέραιοι N και M
- γραμμές $\mathbf{2 + i}$ ($0 \le i \le N$-1): P_i
- γραμμή $N + 2$: Q – το πλήθος των κλήσεων της `minLength`
- γραμμή $\mathbf{N + 3 + i}$ ($0 \le i \le Q$-1): τους αριθμούς D – παράμετροι για κάθε κλήση της `minLength`

Το Lgrader θα τυπώσει Q αριθμούς, έναν για κάθε γραμμή, που αντιστοιχούν στις τιμές επιστροφής των κλήσεων της `minLength`.

Παράδειγμα τοπικού *ελέγχου*

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

HAPPINESS

Το νομισματικό σύστημα στη Χ-χώρα είναι κάπως περίεργο. Υπάρχουν χαρτονομίσματα με αξίες από όλο το εύρος των ακεραίων από 1 έως και *M*. Υπάρχει άλλος ένας περίεργος νόμος στα μαγαζιά της Χ-χώρας – ο πελάτης δεν μπορεί ποτέ να πάρει ρέστα, αλλά επίσης δεν μπορεί να αφήσει φιλοδώρημα – με άλλα λόγια ο πελάτης πρέπει πάντα να πληρώνει το ακριβές αντίτιμο των αγορών του. Αν δεν έχει το ακριβές άθροισμα για την αγορά του, τότε δεν μπορεί να την πραγματοποιήσει. Φανταστείτε τι αναστάτωση δημιουργεί αυτό για τους πελάτες.

Η Niya είναι ένα κορίτσι από τη Χ-χώρα. Σαν όλα τα άλλα άτομα, συνεχώς έρχεται αντιμέτωπη με τους κανόνες που περιγράφηκαν παραπάνω. Πάντοτε γνωρίζει ποια χαρτονομίσματα έχει – ας υποθέσουμε ότι οι αξίες τους είναι $a_1, a_2,\ldots,a_N$.

Όλες αυτές οι τιμές είναι μεταξύ 1 και *M* και μπορεί να κατέχει περισσότερο από ένα χαρτονόμισματα από κάθε είδος. Επίσης η ακολουθία των τιμών $a_1, a_2,\ldots\ldots,a_N$, δεν είναι ταξινομημένη με κανένα τρόπο. Η Niya νιώθει χαρούμενη, όταν μπαίνει σε ένα μαγαζί, αν μπορεί να αγοράσει οποιοδήποτε συνδυασμό από προϊόντα με συνολική αξία ίση με οποιοδήποτε αριθμό μεταξύ 1 και του συνολικού αθροίσματος των χαρτονομισμάτων της $a_1 + a_2 + ... + a_N$. Σε αυτή την περίπτωση, όταν ψωνίζει, το μόνο για το οποίο έχει να νοιαστεί είναι το συνολικό ποσό των χρημάτων, χωρίς να κάνει πολύπλοκους υπολογισμούς για το αν μπορεί (ή όχι) να αγοράσει κάτι με τα χαρτονομίσματά της.

Παρατήρηση: Ας ταξινομήσουμε τα $a_1, a_2,\ldots\ldots,a_N$ σε αύξουσα σειρά. Ας θεωρήσουμε $S_i = 1+a_1+ a_2+\ldots\ldots+a_i$. Απαραίτητη και ικανή συνθήκη για να μπορέσουμε να αναπαραστήσουμε κάθε αριθμό από το 1 μέχρι και το $a_1+ a_2+\ldots\ldots+a_N$ ως άθροισμα στοιχείων από το πολυσύνολο $a_1, a_2,\ldots\ldots,a_N$, είναι η ανισότητα $S_i \geq a_{i+1}$ να ισχύει για κάθε *$i>1$ και να έχουμε $a_1 = 1$*.

Όπως είναι αναμενόμενο, το πολυσύνολο των χαρτονομισμάτων της Niya αλλάζει μετά από κάθε αγορά και επίσης μετά από κάθε μισθό που λαμβάνει – γι' αυτό το λόγο η ευτυχία της είναι μεταβαλλόμενη.

Μπορείτε να βοηθήσετε το κορίτσι με ένα πρόγραμμα. Το πρόγραμμά σας θα δέχεται σαν είσοδο το αρχικό πολυσύνολο των χαρτονομισμάτων της Niya και όλα τα γεγονότα που συμβαίνουν – αγορές και μισθοί. Το πρόγραμμα πρέπει να μπορεί να αποφασίσει αν η Niya είναι ευτυχισμένη στην αρχή και μετά από κάθε γεγονός.

Πρέπει να παρατηρήσουμε ότι η Niya νιώθει ευτυχισμένη επίσης όταν δεν έχει καθόλου χρήματα – τότε απλά αποφεύγει να πάει ψώνια και πάει για τρέξιμο.

Πρόβλημα

Να υλοποιήσετε τις συναρτήσεις *init()* και *is_happy()*, οι οποίες θα μεταγλωττιστούν με τον grader της επιτροπής. Αυτές οι συναρτήσεις πρέπει να εξυπηρετούν τον προσδιορισμό της ευτυχίας της Niya στην αρχή και μετά από κάθε γεγονός. Οι συναρτήσεις θα λαμβάνουν ως παραμέτρους το αρχικό πολυσύνολο των χαρτονομισμάτων και τα πολυσύνολα των χαρτονομισμάτων που αφαιρούνται από το πολυσύνολο (κατά τις αγορές) και αυτών που προστίθενται στο πολυσύνολο (κατά τη λήψη μισθού).

Λεπτομέρειες υλοποίησης

Πρέπει να υποβάλετε στο σύστημα του grader ένα αρχείο **happiness.cpp,** το οποίο περιέχει τις συναρτήσεις:

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

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

Περιγραφή παραμέτρων:

coinsCount – το πλήθος των χαρτονομισμάτων, τα οποία λαμβάνονται (αρχικό πολυσύνολο ή μισθός) ή αφαιρούνται (αγορές).

maxCoinSize – η μέγιστη αξία ενός χαρτονομίσματος.

coins[] – ένας πίνακας, στον οποίο δίνονται με τυχαία σειρά οι αξίες των χαρτονομισμάτων (το index ξεκινάει από το 0).

event – ο τύπος του γεγονότος:

−1 – Αγορά.
1 – Μισθός.

Η συνάρτηση *init* καλείται μία φορά από τον grader στην αρχή για να θέσει το αρχικό πολυσύνολο των χαρτονομισμάτων της Niya και στη συνέχεια ο grader καλεί *Q* φορές τη συνάρτηση *is_happy* με *event* = −1 (αγορά) ή *event* = 1 (μισθός). Μετά από κάθε κλήση η κληθείσα συνάρτηση πρέπει να επιστρέφει *true*, αν η Niya νιώθει ευτυχισμένη με το τρέχον πολυσύνολο χαρτονομισμάτων της, ή *false* σε αντίθετη περίπτωση.

Το αρχείο **happiness.cpp** ΔΕΝ πρέπει να περιέχει τη συνάρτηση *main()*, αλλά μπορεί να περιέχει άλλες δηλώσεις και συναρτήσεις, απαραίτητες για τη σωστή λειτουργία των συναρτήσεων *init* και *is_happy*. Το πρόγραμμά σας πρέπει να περιέχει στην αρχή το `#include "happiness.h"`

Περιορισμοί

Έστω *Nc* το πλήθος των χαρτονομισμάτων της Niya σε οποιαδήποτε δεδομένη στιγμή και *K* – το πλήθος των χαρτονομισμάτων που χρησιμοποιούνται σε οποιαδήποτε αγορά η λήψη μισθού. Τότε έχουμε:

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

Εξασφαλίζεται ότι σε κάθε κλήση της *is_happy* με *event* = −1 (αγορά) το πολυσύνολο των χαρτονομισμάτων που δίνονται στο *coins*[] είναι υποσύνολο του τρέχοντος πολυσυνόλου των χαρτονομισμάτων της Niya.

Παράδειγμα

Καλούμενη συνάρτηση	event	coinsCount	maxCoinSize	coins[]	Τιμή επιστροφής
init		5	100	4 8 1 2 16	true
is_happy	−1(αγορά)	2		2 8	false
is_happy	1(μισθός)	3		7 9 2	true

Υποπροβλήματα

Υποπρό-βλημα	Μονάδες	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

Τοπικός έλεγχος

Για να μπορέσετε να ελέγξετε τις συναρτήσεις σας *init* και *is_happy* στον τοπικό σας υπολογιστή, σας δίνονται τα αρχεία *lgrader.cpp* και *happiness.h*. Μεταγλωττίστε τα μαζί με το αρχείο σας **happiness.cpp** και θα λάβετε ένα πρόγραμμα, το οποίο μπορείτε να χρησιμοποιήσετε για να ελέγξετε τις συναρτήσεις σας.

Το πρόγραμμα απαιτεί την παρακάτω μορφή εισόδου:

Δύο θετικοί ακέραιοι *N* και *M* δίνονται στην πρώτη γραμμή – το αρχικό πλήθος χαρτονομισμάτων της Niya και η μέγιστη αξία που μπορεί να έχει ένα χαρτονόμισμα.

N θετικοί ακέραιοι δίνονται στη δεύτερη σειρά, χωρισμένοι με κενά – οι αξίες των χαρτονομισμάτων στο αρχικό πολυσύνολο.

Στην τρίτη σειρά δίνεται ένας μη αρνητικός ακέραιος *Q* – το πλήθος των γεγονότων.

Σε κάθε μία από τις επόμενες *Q* σειρές περιγράφεται ένα γεγονός – πρώτα, δίνεται μία τιμή που αντιστοιχεί στο είδος του γεγονότος: −1 (αγορά) ή 1 (μισθός). Μετά από αυτό δίνεται ένας θετικός ακέραιος *K* – το πλήθος των χαρτονομισμάτων που αφαιρούνται ή προστίθενται στο πολυσύνολο της Niya. Τέλος, δίνονται *K* ακέραιοι, χωρισμένοι με κενά – οι αξίες των χαρτονομισμάτων που αφαιρούνται ή προστίθενται.

Το πρόγραμμα θα τυπώνει στο standard output *Q*+1 γραμμές με 0 ή 1 – τις καταστάσεις της "ευτυχίας" της Niya στην αρχή και μετά από κάθε γεγονός.

Παράδειγμα για τοπικό έλεγχο

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

Ultimate TTT

Το **Tic-Tac-Toe** είναι ένα παιχνίδι για δύο άτομα με πολύ απλούς κανόνες. Το παιχνίδι παίζεται από ένα (αρχικά άδειο) πίνακα με τρεις γραμμές και τρεις στήλες. Ο πρώτος παίκτης διαλέγει ένα από τα κελιά και βάζει το **σήμα** του ('Χ'). Μετά ο δεύτερος παίκτης διαλέγει ένα άδειο κελί και βάζει το δικό του **σήμα** ('Ο'). Στη συνέχεια ο πρώτος παίκτης ('Χ') διαλέγει ξανά ένα άδειο κελί κλπ. Οι παίκτες παίζουν με τη σειρά και δεν μπορούν να αλλάξουν σειρά (φυσικά δεν κάνει νόημα να το κάνουν αυτό). Το παιχνίδι συνεχίζεται μέχρι ένας από τους δύο παίκτες έχει τρία συνεχόμενα σήματα στην ίδια γραμμή, στήλη ή μια από τις δύο διαγωνίους. Αν ο πίνακας γεμίσει χωρίς κανείς από τους παίκτες να κερδίσει, το παιχνίδι θεωρείται ισόπαλο.

Η Έλλη και ο Χρίστος παίζουν μια τροποποιημένη έκδοση του παιχνιδιού αυτού. Αντί για τον κλασσικό πίνακα 3-επί-3, ξεκινούν με έναν πίνακα άπειρου μεγέθους. Μετά την κίνηση του πρώτου παίκτη (δεν έχει σημασία σε ποια θέση στον πίνακα), η επόμενη κίνηση μπορεί να γίνει σε ένα κελί, το οποίο πρέπει να είναι σε ένα πίνακα 3-επί-3 σε σχέση με την προηγούμενη κίνηση. Με άλλα λόγια, η δεύτερη κίνηση πρέπει να είναι το πολύ 2 στήλες και το πολύ δύο γραμμές μακριά από την πρώτη κίνηση. Οι επόμενες κινήσεις ακολουθούν την ίδια λογική: οι παίκτες πρέπει πάντα να κάνουν τέτοια επιλογή άδειου κελιού ούτως ώστε όλα τα σημειωμένα κελιά να χωράνε σε ένα πίνακα 3-επί-3. Σημειώστε ότι όσο προχωρά το παιχνίδι, γίνονται λιγότερα τα έγκυρα κελιά στον πίνακα, μέχρι να καταλήξει σε έναν κλασσικό πίνακα 3-επί-3 (υποθέτοντας ότι κανένας από τους παίκτες δεν θα κερδίσει προηγουμένως). Δείτε το παράδειγμα για επεξήγηση.

Όπως και στο κλασσικό παιχνίδι, νικητής είναι ο παίκτης που πρώτος έχει τρία γειτονικά κελιά στην ίδια γραμμή, στην ίδια στήλη ή διαγώνια. Αν όλα τα έγκυρα κελιά γεμίσουν πριν κάποιος κερδίσει, το παιχνίδι θεωρείται ισόπαλο.

Λάβετε υπόψην σας το ακόλουθο παράδειγμα:

. . . .	. . . . 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

Στο παράδειγμα ο πρώτος παίκτης ('X') κέρδισε, αλλά αν ο δεύτερος παίκτης είχε καλύτερη στρατηγική, δεν θα πετύχαινε κάτι τέτοιο.

Θα ονομάσουμε *ιδανικό παιχνίδι* τη δημιουργία τέτοιων κινήσεων, όπου ο παίκτης κερδίζει αν το παιχνίδι μπορεί να κερδιθεί, έρχεται ισόπαλος αν το παιχνίδι δεν μπορεί να κερδιθεί, αλλά μπορεί να έρθει ισόπαλο και χάνει αν καμιά από τις προηγούμενες επιλογές δεν είναι διαθέσιμη. Υποθέτουμε ότι ο αντίπαλος παίζει με τον ίδιο τρόπο.

Πρόβλημα

Γράψε ένα πρόγραμμα το οποίο, εάν δίνεται η παρούσα κατάσταση του πίνακα βρίσκει ποιο κελί πρέπει να επιλέξει ο παίκτης αν παίζει ιδανικά.

Είσοδος

Η πρώτη γραμμή της κανονικής εισόδου έχει δύο ακεραίους **N** και **M** – που είναι ο αριθμός των απομείναντων έγκυρων γραμμών και στηλών αντίστοιχα. Κάθε μια από τις ακόλουθες **N** γραμμές περιέχει μια συμβολοσειρά μήκους **M**, που περιγράφει την επόμενη γραμμή του πίνακα. Όλες οι γραμμές δημιουργούν μια σωστή περιγραφή της παρούσας κατάστασης του πίνακα. Είναι εγγυημένο ότι τουλάχιστον μια κίνηση έχει ήδη γίνει (άρα τα υπόλοιπα έγκυρα κελιά είναι πεπερασμένα) και ότι το παιχνίδι δεν έχει ακόμη τελειώσει.

Έξοδος

Το πρόγραμμα τυπώνει σε μια γραμμή της κανονικής εξόδου δύο αριθμούς **R** και **C** με κενό μεταξύ τους, που αντιστοιχούν στον αριθμό της γραμμής και της στήλης (μετρούμε από το 1). Οι αριθμοί αυτοί είναι οι συντεταγμένες του κελιού, το οποίο ο παίκτης θα διαλέξει για να παίξει ιδανικά. Αν υπάρχουν περισσότερες επιλογές, η έξοδος σας μπορεί να είναι οποιαδήποτε από αυτές.

Περιορισμοί

- 3 ≤ **N**, **M** ≤ 5
- Όλα τα σύμβολα που περιγράφουν τον πίνακα είναι από το αλφάβητο {'.' (τελεία), 'X' (στα κεφαλαία), 'O' (στα κεφαλαία)}.
- Σε test cases, που αξίζουν 50% των βαθμών για την εργασία έχουν πίνακα μεγέθους, **N** = **M** = 3 (με άλλα λόγια ο πίνακας είναι το κλασσικό παιχνίδι Tic-Tac-Toe (τρίλιζα).

Αξιολόγηση

Τα προβλήματα χωρίζονται σε 4 υπο-προβλήματα (subtasks). Κάθε υπο-πρόβλημα παίρνει 25 βαθμούς, αν όλα τα ερωτήματα του υπο-προβλήματος απαντηθούν σωστά.

Παράδειγμα

Είσοδος	Έξοδος
3 4 .XX. .... .O..	1 1

Επεξήγηση

Είναι η σειρά του δεύτερου παίκτη (‘O’) να παίξει. Αν διαλέξει τη θέση (1, 4), τότε ο ‘X’ θα νικήσει αν παίξει στη θέση (2, 3) – ανεξάρτητα από ποιο κελί θα επιλέξει ο παίκτης (‘O’), ο ‘X’ θα έχει πάντα νικητήρια κίνηση.

Όμως αν ο δεύτερος παίκτης (‘O’) παίξει στη θέση (1, 1) η επόμενη κίνηση θα περιοριστεί στις στήλες 1..3 του πίνακα. ΄Ετσι θα είναι αδύνατο για τον ‘X’ να κερδίσει σε μια κίνηση παίζοντας στη θέση (1, 4). Με τη χρήση αυτής της κίνησης υπάρχει στρατηγική του ‘O’ που θα οδηγήσει το παιχνίδι σε ισοπαλία.