

## SmartRuspa (ruspa)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

William è stato incaricato di abbattere una serie di  $N$  muri disposti lungo una linea per fare posto a una nuova strada. Non avendo la patente, per farlo si è procurato un nuovissimo esemplare di *SmartRuspa* in grado di svolgere tutto il lavoro da sola! In particolare, la *SmartRuspa* è stata posizionata dagli addetti in una certa coordinata  $X$  con un certo orientamento, verso destra (coordinate crescenti) o verso sinistra (coordinate decrescenti). Da lì la ruspa parte e procede fino a quando non incontra un muro da abbattere. Dopo aver abbattuto un muro, inverte automaticamente la marcia e prosegue nel suo algoritmo.

Informatosi sui blog competenti, William è conscio del difetto di queste ruspe di partire ad un certo punto in una direzione verso l'infinito (o peggio, verso la casa del vicino). Per prevenire onerose richieste di danni, vuole quindi trovarsi pronto a saltare sulla ruspa per spegnerla non appena vede che sta per uscire dai confini prestabiliti. Deve tuttavia decidere se posizionarsi sul confine di destra o in quello di sinistra. Aiuta William calcolando da quale direzione la ruspa uscirà al termine del suo algoritmo!

## Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

📎 Tra gli allegati a questo task troverai un template (`ruspa.c`, `ruspa.cpp`, `ruspa.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare le seguenti funzioni:

C/C++	<code>void inizia(int N, int M[]);</code>
Pascal	<code>procedure inizia(N: longint; var M: array of longint);</code>

In cui:

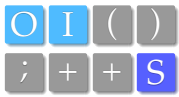
- L'intero  $N$  rappresenta il numero di muri.
- L'array  $M$ , indicizzato da 0 a  $N - 1$ , contiene le coordinate dei muri in ordine crescente.

C/C++	<code>int abbatti(int X, int D);</code>
Pascal	<code>function abbatti(X, D: longint): longint;</code>

In cui:

- L'intero  $X$  rappresenta la coordinata in cui la ruspa è stata posizionata.
- L'intero  $D$  rappresenta la direzione (o l'orientamento) verso cui la ruspa è stata posizionata: vale 0 se è verso destra e 1 se è verso sinistra.
- La funzione dovrà restituire l'orientamento finale della ruspa (secondo la medesima convenzione precedente), che verrà stampato sul file di output.

Il template chiamerà prima la procedura `inizia`, e poi  $T$  volte la funzione `abbatti` di cui stamperà l'output. Ogni esecuzione di `abbatti` si svolge sugli stessi muri indipendentemente dalle precedenti (i muri abbattuti durante la prima esecuzione non sono da contare come già abbattuti per la seconda).



## Dati di input

Il file `input.txt` è composto da  $T + 2$  righe. La prima riga contiene i due interi  $N$  e  $T$ . La seconda riga contiene gli  $N$  interi  $M_i$  separati da uno spazio. Le successive  $T$  righe contengono ciascuna due interi  $X$  e  $D$ , che descrivono una chiamata ad **abbatti**.

## Dati di output

Il file `output.txt` è composto da un'unica riga contenente  $T$  valori 0 (se la ruspa termina verso destra) o 1 (se termina verso sinistra).

## Assunzioni

- $1 \leq N, T \leq 100\,000$ .
- $-1\,000\,000 \leq M_i \leq 1\,000\,000$  per ogni  $i = 0 \dots N - 1$  e sono in ordine *strettamente* crescente.
- $0 \leq D \leq 1$  per ogni chiamata ad **abbatti**.
- $-1\,000\,000 \leq X \leq 1\,000\,000$  non coincide mai con la posizione di un muro.

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]**: Casi d'esempio.
- **Subtask 2 [20 punti]**:  $N, T \leq 10$ .
- **Subtask 3 [40 punti]**:  $N \leq 10\,000, T \leq 100$ .
- **Subtask 4 [30 punti]**: Nessuna limitazione specifica.

## Esempi di input/output

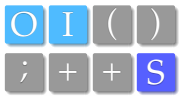
input.txt	output.txt
3 1 -10 0 20 10 0	0
6 3 -3 -1 5 7 15 42 6 0 6 1 10 1	0 1 0



## Spiegazione

Nel **primo caso di esempio**, la ruspa abbatte prima il muro in posizione 20, poi quello in posizione 0 e infine esce sulla destra.

Nel **secondo caso di esempio**, nei primi due test la ruspa abbatte tutti i muri uscendo nella direzione con cui era partita. Nell'ultimo test, abbatte per ultimo il muro in posizione  $-1$  uscendo a destra e lasciando in piedi il muro in posizione  $-3$ .



## Pausa caffè (caffè)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

Durante i lunghi ed estenuanti stage di preparazione dei PO (probabili olimpici), gli  $N$  tutor ogni tanto recuperano le energie andando a prendersi un caffè. Per ottimizzare la loro presenza in aula, ogni tutor fa pausa in un momento diverso del giorno: in particolare, il tutor  $i$ -esimo fa pausa nell'intervallo di tempo  $[A_i, B_i]$  dove i valori  $A_i$  e  $B_i$  sono tutti distinti tra loro.

I tutor tra di loro sono sempre molto cordiali: ogni volta che uno di loro raggiunge la macchinetta del caffè, offre sempre un giro di caffè a tutti i presenti. Questo elevato consumo di caffè sta facendo rapidamente finire le scorte a disposizione della macchinetta. Aiuta Monica a programmare i rifornimenti calcolando quanti caffè vengono bevuti ogni giorno!

## Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

☞ Tra gli allegati a questo task troverai un template (`caffè.c`, `caffè.cpp`, `caffè.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int pausa(int N, int A[], int B[]);</code>
Pascal	<code>function pausa(N: longint; var A, B: array of longint): longint;</code>

In cui:

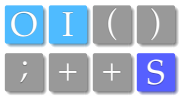
- L'intero  $N$  rappresenta il numero di tutor.
- Gli array  $A$  e  $B$ , indicizzati da  $0$  a  $N - 1$ , descrivono gli intervalli  $[A_i, B_i]$  di tempo in cui i tutor vanno a prendersi il caffè.
- La funzione dovrà restituire il numero totale di caffè consumati, che verrà stampato sul file di output.

## Dati di input

Il file `input.txt` è composto da  $N + 1$  righe. La prima riga contiene l'unico intero  $N$ . Le successive  $N$  righe contengono ciascuna due interi  $A_i, B_i$  separati da uno spazio.

## Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.



## Assunzioni

- $1 \leq N \leq 40\,000$ .
- $0 \leq A_i < B_i \leq 1\,000\,000$  per ogni  $i = 0 \dots N - 1$ .
- I valori  $A_i$  e  $B_i$  sono tutti differenti tra loro.

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]:** Casi d'esempio.
- **Subtask 2 [20 punti]:**  $N \leq 10$ .
- **Subtask 3 [40 punti]:**  $N \leq 1000$ .
- **Subtask 4 [30 punti]:** Nessuna limitazione specifica.

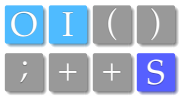
## Esempi di input/output

input.txt	output.txt
3 4 7 5 10 1 6	6
6 1 6 4 8 7 9 14 17 10 12 13 20	9

## Spiegazione

Nel **primo caso di esempio**, prima arriva il tutor 2 che si prende il suo caffè, poi il tutor 0 che lo offre al tutor 2 e infine il tutor 1 che lo offre a entrambi gli altri per un totale di  $1 + 2 + 3 = 6$  caffè.

Nel **secondo caso di esempio**, il tutor 1 offre il caffè al tutor 0, il tutor 2 lo offre al tutor 1, il tutor 3 lo offre al tutor 5 per un totale di  $1 + 2 + 2 + 2 + 1 + 1 = 9$  caffè.



## Regali gemelli (regali)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

William deve andare a comprare un regalo di compleanno per i suoi due giovani cuginetti. Essendo fratelli gemelli, sa che se gli facesse due regali troppo diversi sicuramente inizierebbero a litigare. Pertanto si è procurato un listino di un negozio contenente la descrizione di  $N$  giocattoli, ciascuno individuato da  $Q$  qualità (rappresentate con dei numeri da 1 a 1000).

Dato che le qualità dei giocattoli sono elencate in ordine di importanza e il significato di una qualità dipende dalle precedenti, la simiglianza di due giocattoli è definita come *il più lungo prefisso comune* tra le loro due descrizioni. Trova i giocattoli più simili tra loro!

### Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

📁 Tra gli allegati a questo task troverai un template (`regali.c`, `regali.cpp`, `regali.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int compra(int N, int Q, int* G[]);</code>
Pascal	<code>function compra(N, Q: longint; var V: matrix): longint;</code>

In cui:

- L'intero  $N$  rappresenta il numero di giocattoli presenti nel catalogo.
- L'intero  $Q$  rappresenta il numero di qualità che descrivono ogni giocattolo.
- L'array bidimensionale  $G$ , indicizzato da  $0$  a  $N - 1$  e da  $0$  a  $Q - 1$ , contiene le qualità dei giocattoli.
- La funzione dovrà restituire la massima simiglianza possibile tra due giocattoli, che verrà stampata sul file di output.

### Dati di input

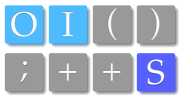
Il file `input.txt` è composto da  $N + 1$  righe. La prima riga contiene i due interi  $N$  e  $Q$ . Le successive  $N$  righe contengono ciascuna  $Q$  interi  $G_{i,j}$  separati da uno spazio.

### Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

### Assunzioni

- $2 \leq N \leq 10\,000$ .
- $1 \leq Q \leq 100$ .
- $1 \leq G_{i,j} \leq 1000$  per ogni  $i = 0 \dots N - 1$ ,  $j = 0 \dots Q - 1$ .



## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]:** Casi d'esempio.
- **Subtask 2 [20 punti]:**  $N, Q \leq 10$ .
- **Subtask 3 [40 punti]:**  $N \leq 100$ .
- **Subtask 4 [30 punti]:** Nessuna limitazione specifica.

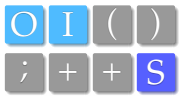
## Esempi di input/output

input.txt	output.txt
3 7 10 1 7 5 4 2 3 10 1 7 5 9 8 6 10 1 7 5 1 1 1	4
5 5 2 1 4 1 2 1 2 3 4 5 4 2 3 4 5 2 1 3 3 1 2 1 4 4 3	3

## Spiegazione

Nel **primo caso di esempio**, ogni coppia di giocattoli ha esattamente le prime quattro qualità in comune.

Nel **secondo caso di esempio**, il primo e l'ultimo giocattolo hanno le prime tre qualità in comune e sono i più simili.



## Giro in bici (ciclismo)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

Giorgio è appassionato di ciclismo, tanto che ogni giorno prende un treno per raggiungere una nuova località e da lì farsi un bel giro in bici. La mappa della località in cui si è recato oggi consiste di  $N$  incroci collegati da  $M$  strade bidirezionali (non esistono sensi unici per le biciclette). Ognuno degli incroci, inoltre, si trova a una diversa altitudine  $H_i$ .

Nel suo giro, Giorgio partirà dalla stazione (corrispondente all'incrocio numero 0) e procederà ogni volta dirigendosi verso l'incrocio con altitudine più bassa tra quelli collegati all'incrocio corrente (Giorgio non ama le salite!), eccettuato quello da cui al momento arriva (in altre parole, non fa mai inversione a U). Il giro in bici prosegue quindi fintanto che si troverebbe costretto a effettuare un'inversione a U oppure quando ritorna in un incrocio da cui è già passato. In quale incrocio finirà il giro in bici?

## Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

📄 Tra gli allegati a questo task troverai un template (`ciclismo.c`, `ciclismo.cpp`, `ciclismo.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int pedala(int N, int M, int H[], da[], int a[]);</code>
Pascal	<code>function pedala(N, M: longint; var H, da, a: array of longint): longint;</code>

In cui:

- L'intero  $N$  rappresenta il numero di incroci presenti.
- L'intero  $M$  rappresenta il numero di strade presenti.
- L'array  $H$ , indicizzato da 0 a  $N - 1$ , contiene le altitudini degli incroci.
- Gli array  $da$  e  $a$ , indicizzati da 0 a  $M - 1$ , contengono la descrizione delle strade (per cui la strada  $i$ -esima collega gli incroci  $da[i]$  e  $a[i]$ ).
- La funzione dovrà restituire l'incrocio in cui termina il giro in bici, che verrà stampato sul file di output.

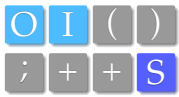
## Dati di input

Il file `input.txt` è composto da  $M + 2$  righe. La prima riga contiene i due interi  $N$  e  $M$ . La seconda riga contiene gli  $N$  interi  $H_i$  separati da uno spazio. Le successive  $M$  righe contengono ciascuna i due interi  $da[i]$  e  $a[i]$ .

## Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.





## Assunzioni

- $2 \leq N \leq 10\,000$ .
- $1 \leq M \leq 100\,000$ .
- $0 \leq H_i \leq 1\,000\,000$  per ogni  $i = 0 \dots N - 1$ .
- $0 \leq da_i, a_i < N$  e  $da_i \neq a_i$  per ogni  $i = 0 \dots M - 1$ .
- Le altitudini sono tutte distinte e non ci sono strade ripetute.

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]:** Casi d'esempio.
- **Subtask 2 [20 punti]:**  $N \leq 10$ .
- **Subtask 3 [40 punti]:**  $N \leq 100$ .
- **Subtask 4 [30 punti]:** Nessuna limitazione specifica.

## Esempi di input/output

input.txt	output.txt
5 6 20 10 40 5 33 0 1 1 2 0 4 2 3 3 1 4 3	0
6 6 21 17 19 13 43 18 0 1 1 2 0 3 1 4 3 4 4 5	2

## Spiegazione

Nel **primo caso di esempio**, il giro percorre gli incroci  $0 - 1 - 3 - 4 - 0$  e poi termina (dato che ha chiuso un ciclo).

Nel **secondo caso di esempio**, il giro percorre gli incroci  $0 - 3 - 4 - 1 - 2$  e poi termina (dato che non può più muoversi).



## Parser HTML (html)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

William sta cercando di scrivere un parser HTML facendo uso di espressioni regolari (regex), così da evocare Zalgo. Per fortuna questo parser non deve davvero essere in grado di consumare HTML, bensì è sufficiente che sia in grado di effettuare il cosiddetto “unescape”: l’operazione inversa dell’escape.

L’escape è una tecnica che viene utilizzata per rappresentare caratteri che altrimenti avrebbero un significato specifico nel linguaggio utilizzato. Per questa particolare applicazione, William ha bisogno di effettuare l’unescape soltanto del carattere `&` che, quando viene normalmente sottoposto all’escape in HTML, diventa `&amp;`; passando quindi da 1 carattere a 5 caratteri.

William ha notato però una cosa: una volta completato l’unescape di un intero file HTML, potrebbe essere possibile *eseguire di nuovo l’unescape sullo stesso file*, dato che nel frattempo il codice HTML è cambiato e potrebbero essere comparse altre entità `&amp;`; da convertire.

Dato il file HTML, calcola il contenuto del file HTML alla fine di questa serie di lanci del parser.

### Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

🔍 Tra gli allegati a questo task troverai un template (`html.c`, `html.cpp`, `html.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int unescape(int N, char S[]);</code>
Pascal	<code>function unescape(N: longint; var S: array of char): longint;</code>

In cui:

- L’intero  $N$  rappresenta la lunghezza (in caratteri) del file HTML.
- L’array  $S$ , indicizzato da 0 a  $N - 1$ , contiene i singoli caratteri che compongono il file HTML.
- La funzione deve modificare l’array  $S$  in modo che rifletta il contenuto finale del file HTML.
- La funzione deve inoltre restituire un intero: la lunghezza del file HTML finale.

### Dati di input

Il file `input.txt` può contenere diverse righe. La prima riga contiene il numero intero  $N$ . Dalla riga seguente, si trovano  $N$  caratteri che rappresentano il contenuto del file HTML.

### Dati di output

Il file `output.txt` è composto dallo stesso numero di righe presenti in `input`. La prima riga contiene il numero di caratteri (minore o uguale a  $N$ ) che compongono il file HTML al termine delle varie conversioni. Dalla seconda riga in poi saranno stampati gli  $N$  caratteri che formano effettivamente il file.



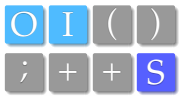
## Assunzioni

- $2 \leq N \leq 1\,000\,000$ .
- $S$  è formata da caratteri ASCII stampabili (a parte l'“a capo”) per ogni  $i = 0 \dots N - 1$ . Non è detto che siano presenti i tag, o meglio:  $S$  potrebbe avere un contenuto *non valido* come HTML.
- Il file termina (come sempre) con un carattere “a capo”, che è però compreso negli  $N$  caratteri totali.

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]**: Casi d'esempio.
- **Subtask 2 [20 punti]**:  $N \leq 500$ .
- **Subtask 3 [40 punti]**:  $N \leq 10\,000$ .
- **Subtask 4 [30 punti]**: Nessuna limitazione specifica.



## Esempi di input/output

input.txt	output.txt
<pre>77 &lt;a href="https://youtu.be/ dQw4w9WgXcQ"&gt;   Clicca &amp; vinci un iPad! &lt;/a&gt;</pre>	<pre>73 &lt;a href="https://youtu.be/ dQw4w9WgXcQ"&gt;   Clicca &amp; vinci un iPad! &lt;/a&gt;</pre>
<pre>87 &lt;code&gt; if (a == 0 &amp;&amp;&amp; b == 1) {   c  = 13;   c &amp;&amp;= 31; } &lt;/code&gt;</pre>	<pre>67 &lt;code&gt; if (a == 0 &amp;&amp; b == 1) {   c  = 13;   c &amp;= 31; } &lt;/code&gt;</pre>
<pre>168 &lt;html&gt;   &lt;head&gt;     &lt;title&gt;Hello world!&lt;/title&gt;   &lt;/head&gt;   &lt;body&gt;     Hello &amp;&amp;&amp; world!     Tizio &amp; amp; caio!     HTML &amp;&amp; co :)   &lt;/body&gt; &lt;/html&gt;</pre>	<pre>144 &lt;html&gt;   &lt;head&gt;     &lt;title&gt;Hello world!&lt;/title&gt;   &lt;/head&gt;   &lt;body&gt;     Hello &amp; world!     Tizio &amp; amp; caio!     HTML &amp; co :)   &lt;/body&gt; &lt;/html&gt;</pre>

## Robot programmabile (programma)

Limite di tempo: 1.0 secondi  
 Limite di memoria: 256 MiB

William ha appena costruito un nuovo robot programmabile per il suo laboratorio di robotica, in grado di rispondere a ben cinque comandi *di basso livello*:

- 'R' (right): gira di  $90^\circ$  a destra;
- 'L' (left): gira di  $90^\circ$  a sinistra;
- 'F' (forward): avanza di 1 nella direzione in cui è orientato;
- 'B' (backward): indietreggia di 1 rispetto alla direzione in cui è orientato;
- 'X' (explode): attiva l'autodistruzione ed esplode.

William ha dunque scritto un lungo programma  $P$  composto di  $N$  comandi e lo ha memorizzato nel robot. In questo modo, il robot può essere comandato con comandi *di alto livello* ( $S : E$ ) tali che  $0 \leq S \leq E < N$ . Ciascuno di questi comandi verrà convertito dal robot nella sequenza di comandi di basso livello  $P_S, P_{S+1}, \dots, P_E$  ed eseguito in questo modo.

William ha appena inviato una sequenza di  $M$  comandi di alto livello al robot, posizionato inizialmente nel quadretto  $(0, 0)$  e orientato verso est (coordinate  $x$  crescenti). Il robot ora eseguirà la sequenza fino alla fine oppure fino al momento in cui esplode. In quale quadretto si troverà alla fine?

### Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

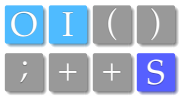
📎 Tra gli allegati a questo task troverai un template (`programma.c`, `programma.cpp`, `programma.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>point passeggia(int N, int M, char P[], int S[], int E[]);</code>
Pascal	<code>function passeggia(N, M: longint; var P: array of char; var S, E: array of longint): point;</code>

In cui:

- L'intero  $N$  rappresenta il numero di caratteri di cui il programma interno  $P$  è composto.
- L'intero  $M$  rappresenta il numero di comandi di alto livello da eseguire.
- L'array  $P$ , indicizzato da  $0$  a  $N - 1$ , contiene la sequenza di caratteri 'RLFBX' che descrivono il programma interno.
- Gli array  $S$  ed  $E$ , indicizzati da  $0$  a  $M - 1$ , contengono la sequenza di comandi di alto livello ( $S_i : E_i$ ) da eseguire.
- La funzione dovrà restituire la posizione finale del robot, che verrà stampata sul file di output.



## Dati di input

Il file `input.txt` è composto da  $M + 2$  righe. La prima riga contiene i due interi  $N$  ed  $M$ . La seconda riga contiene la stringa  $P$  (concatenata senza spazi). Le successive  $M$  righe contengono ciascuna i due interi  $S_i$  ed  $E_i$ .

## Dati di output

Il file `output.txt` è composto da un'unica riga contenente due interi, le coordinate finali del robot.

## Assunzioni

- $1 \leq N \leq 1\,000\,000$ .
- $1 \leq M \leq 100\,000$ .
- $0 \leq S_i \leq E_i < N$  per ogni  $i = 0 \dots M - 1$ .
- $P_i$  è un carattere tra 'RLF BX'.

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]:** Casi d'esempio.
- **Subtask 2 [30 punti]:**  $N, M \leq 100$ .
- **Subtask 3 [20 punti]:** Sono presenti solo comandi 'F' e 'B'.
- **Subtask 4 [20 punti]:** Sono presenti solo comandi 'F', 'B' e 'X'.
- **Subtask 5 [10 punti]:** Non è presente il comando 'X'.
- **Subtask 6 [10 punti]:** Nessuna limitazione specifica.

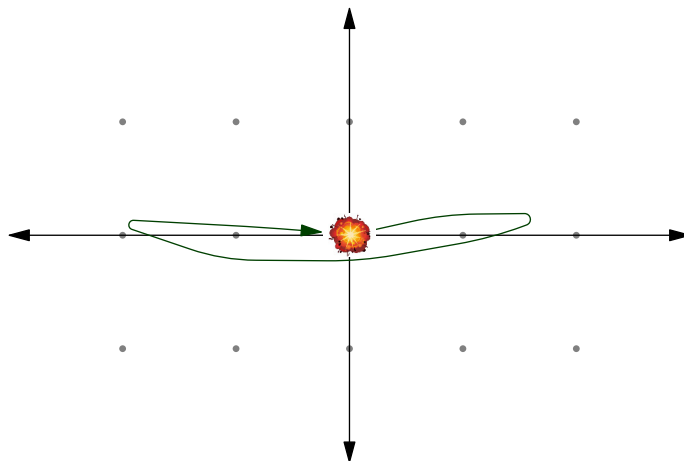
## Esempi di input/output

input.txt	output.txt
13 4 FFFBFFXBBFB 1 3 10 12 4 11 5 7	0 0
9 4 BBRXLFB 4 7 6 8 1 1 0 2	-1 4

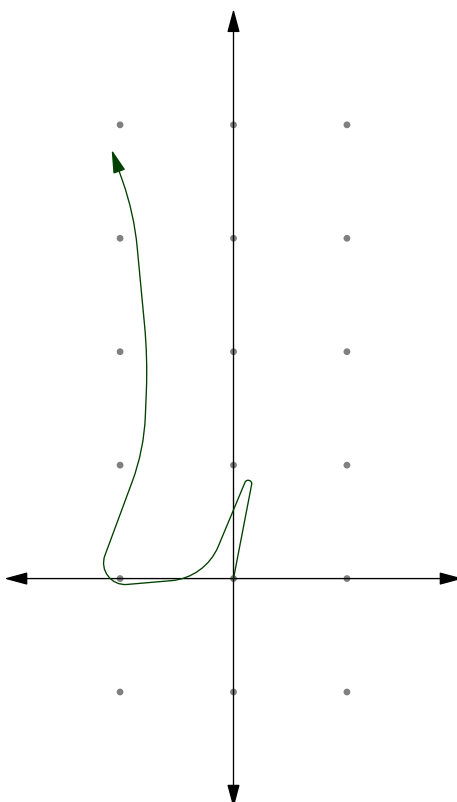


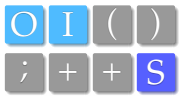
## Spiegazione

Nel **primo caso di esempio**, le istruzioni vengono convertite in FFB BBB FFXBBFBFB FXB e il robot fa il seguente percorso (fino al punto in cui esplode):



Nel **secondo caso di esempio**, le istruzioni vengono convertite in LFBR BRB B BBR e il robot fa il seguente percorso (fino al termine del programma):





## Tracce aliene (tracce)

Limite di tempo: 1.0 secondi  
Limite di memoria: 256 MiB

Giorgio ha deciso di rispondere in modo conclusivo alla domanda che ha assillato l'umanità per tanti anni: esistono oppure no gli alieni? Per questo motivo, si è messo alla ricerca di tracce aliene.

Al fine di rilevare presenze extraterrestri, Giorgio ha eseguito una serie di misurazioni elettromagnetiche. Dopo giorni di misurazioni, è arrivato a determinare con precisione una stringa  $S$  che:

- È composta da sole cifre decimali (da 0 a 9);
- È terminata da una cifra 0 (in ultima posizione);
- Il carattere terminatore 0 è presente solo in ultima posizione e non prima.


Questa stringa potrebbe essere un messaggio degli alieni. Ovviamente però, per verificare la cosa, sarebbe necessario fare un complicatissimo conteggio per sapere quante volte questa stringa compare spontaneamente “in natura”. Per una stima iniziale però, Giorgio farà un conteggio molto semplice.

Fissato un numero  $K$ , Giorgio vuole contare quante sono le sequenze di  $K$  cifre decimali che contengono *almeno una volta* la stringa  $S$  come sottostringa. Per esempio, se  $K$  fosse uguale a 4 e  $S$  uguale a 210, le sequenze da contare sarebbero le seguenti 20:

0210	1210	2100	2101	2102	2103	2104	2105	2106	2107
2108	2109	2210	3210	4210	5210	6210	7210	8210	9210

## Implementazione

Dovrai sottoporre esattamente un file con estensione `.c`, `.cpp` o `.pas`.

 Tra gli allegati a questo task troverai un template (`tracce.c`, `tracce.cpp`, `tracce.pas`) con un esempio di implementazione da completare.

Se sceglierai di utilizzare il template, dovrai implementare la seguente funzione:

C/C++	<code>int conteggio(int K, char S[]);</code>
Pascal	<code>function conteggio(K: longint; var S: ansistring): longint;</code>

In cui:

- $K$  rappresenta il numero di cifre da cui sono formate le sequenze analizzate da Giorgio.
- $S$  è la stringa che Giorgio deve cercare.
- La funzione deve restituire il numero di sequenze contate da Giorgio. Dal momento che questo numero può essere molto grande, è necessario restituirne soltanto il resto della divisione per 1 000 000 007.



☞ Per eseguire l'operazione di modulo si può utilizzare l'operatore `%` del C/C++. In Pascal invece esiste l'operatore `mod`.

Ad esempio, il resto della divisione di 5 per 3 si calcola come `5 % 3` in C/C++ e come `5 mod 3` in Pascal. In entrambi i casi il risultato sarà 2.

L'operazione di modulo, inoltre, ha le seguenti proprietà (molto utili per evitare *integer overflow* quando si vogliono calcolare numeri molto grandi):

- $(A + B) \bmod M = (A \bmod M + B \bmod M) \bmod M$
- $(A \cdot B) \bmod M = (A \bmod M \cdot B \bmod M) \bmod M$

## Dati di input

Il file `input.txt` è composto da due righe. La prima riga contiene l'unico intero  $K$ . La seconda riga contiene la stringa  $S$ .

## Dati di output

Il file `output.txt` è composto da un'unica riga contenente un unico intero, la risposta a questo problema.

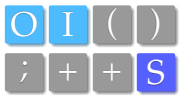
## Assunzioni

- $1 \leq |S| \leq 100\,000$ , dove  $|S|$  è la lunghezza di  $S$ .
- $|S| \leq K \leq 1\,000\,000$ .

## Assegnazione del punteggio

Il tuo programma verrà testato su diversi test case raggruppati in subtask. Per ottenere il punteggio relativo ad un subtask, è necessario risolvere correttamente tutti i test relativi ad esso.

- **Subtask 1 [10 punti]:** Casi d'esempio.
- **Subtask 2 [30 punti]:**  $|S| = 1$ .
- **Subtask 3 [20 punti]:**  $K \leq 6$ .
- **Subtask 4 [20 punti]:**  $K \leq 1000$ .
- **Subtask 5 [20 punti]:** Nessuna limitazione specifica.



## Esempi di input/output

input.txt	output.txt
4 210	20
6 210	3999
210 210	843349294