D. Guessing Game

Nome del problema	Guessing Game
Limite di tempo	4 secondi
Limite di memoria	1 gigaottetto

Nel centro storico di Lund c'è una strada con N case in fila, indicizzate da 0 a $N-1$. In una di queste case vive Emma, i suoi amici Anna e Bertil vogliono capire quale. Invece di limitarsi a dire ai suoi amici dove vive, Emma decide di fare un gioco con loro. Prima dell'inizio del gioco, Anna e Bertil conoscono solo il numero delle case sulla strada. A questo punto, Anna e Bertil possono scegliere un intero positivo K e concordare una strategia. Ogni comunicazione successiva è vietata.

Il gioco consiste in due fasi. Nella prima fase, Emma sceglie un ordine in cui visitare le case, in modo tale che la sua casa sia l'ultima casa visitata. Quindi conduce Anna alle case una per una in questo ordine, senza dirle in anticipo l'ordine. Per ogni casa che non sia la casa di Emma, Anna può scrivere un singolo numero intero compreso tra 1 e K sulla porta d'ingresso della casa con un pezzo di gesso. Per l'ultima casa che visitano (la casa di Emma), Emma scrive lei stessa un numero intero compreso tra 1 e K sulla porta.

Nella seconda fase del gioco, Bertil cammina lungo la strada e legge tutti i numeri scritti sulle porte da Anna ed Emma. Ora vuole indovinare in quale casa vive Emma. Bertil ha **due possibilità** di indovinare correttamente e lui e Anna vincono la partita se ci riesce. Altrimenti, Emma vince la partita.

Riesci a trovare una strategia in cui Anna e Bertil abbiano la garanzia di vincere la partita? La tua strategia verrà valutata in base al valore di K (più piccolo è, meglio è).

Implementazione

Questo è un problema multi-esecuzione, il che significa che il tuo programma verrà eseguito più volte. La prima volta che viene eseguito, implementerà la strategia di Anna. La seconda volta implementerà quella di Bertil.

La prima riga dell'input conterrà due numeri interi, P e N, dove P è 1 o 2 (prima o seconda fase) e N è il numero di case. **Ad eccezione dell'input di esempio (non utilizzato per il punteggio),** N

sarà sempre uguale a $100\,000$.

L'input successivo dipende dalla fase:

Fase 1

Il tuo programma deve iniziare scrivendo il numero K su una singola riga $(1 \leq K \leq 1\,000\,000)$. Quindi, $N-1$ volte, deve leggere una riga contenente un indice i $(0 \leq i < N)$, e restituire una riga con un numero A_i $(1 \leq A_i \leq K)$. Ogni indice i eccetto l'indice della casa di Emma apparirà esattamente una volta, in un ordine deciso dal grader.

Fase 2

Il tuo programma deve leggere una riga con N interi, $A_0, A_1, ..., A_{N-1}$.

Quindi, deve stampare una riga con due numeri interi, s_1 e s_2 $(0 \leq s_i < N)$, gli indici indovinati. s_1 e s_2 possono essere uguali.

Dettagli di implementazione

Nota che quando il tuo programma viene eseguito in Fase 2, il programma è riavviato. Questo vuol dire che non puoi salvare informazioni in variabili tra le esecuzioni.

In entrambe le fasi, assicurati di fare `flush` dello standard output dopo ogni riga stampata, altrimenti il tuo programma potrebbe ricevere il verdetto "Time Limit Exceeded". In Python, `print()` fa `flush` automaticamente. In C++, `cout << endl;` fa `flush` oltre a stampare una nuova riga; se usi `printf`, usa `fflush(stdout)`.

Il grader per questo problema potrebbe essere **adattivo**, nel senso che potrebbe cambiare il suo comportamento a seconda dell'output del tuo programma per impedire a soluzioni euristiche di fare punti. Potrebbe eseguire la fase 1, esaminare l'output e quindi eseguire nuovamente la fase 1 utilizzando le informazioni estratte dalla prima esecuzione.

Il tuo programma *deve* essere deterministico, ovvero comportarsi allo stesso modo se eseguito due volte sullo stesso input. Se desideri utilizzare numeri casuali nel tuo programma, assicurati di utilizzare un `seed` casuale fisso. Questo può essere fatto passando una costante a `srand` (in C++) o `random.seed` (in Python), o, se usi i generatori di numeri casuali di C++11, specificando il seed quando il generatore di numeri casuali viene costruito. In particolare, non puoi usare `srand(time(NULL))` in C++. Se il valutatore rileva che il tuo programma non è deterministico, riceverà un verdetto "Wrong Answer".

Se la *somma* dei tempi di esecuzione di ciascuna delle (fino a 3) esecuzioni separate del tuo programma supera il limite di tempo, la tua soluzione riceverà il verdetto "Time Limit Exceeded".

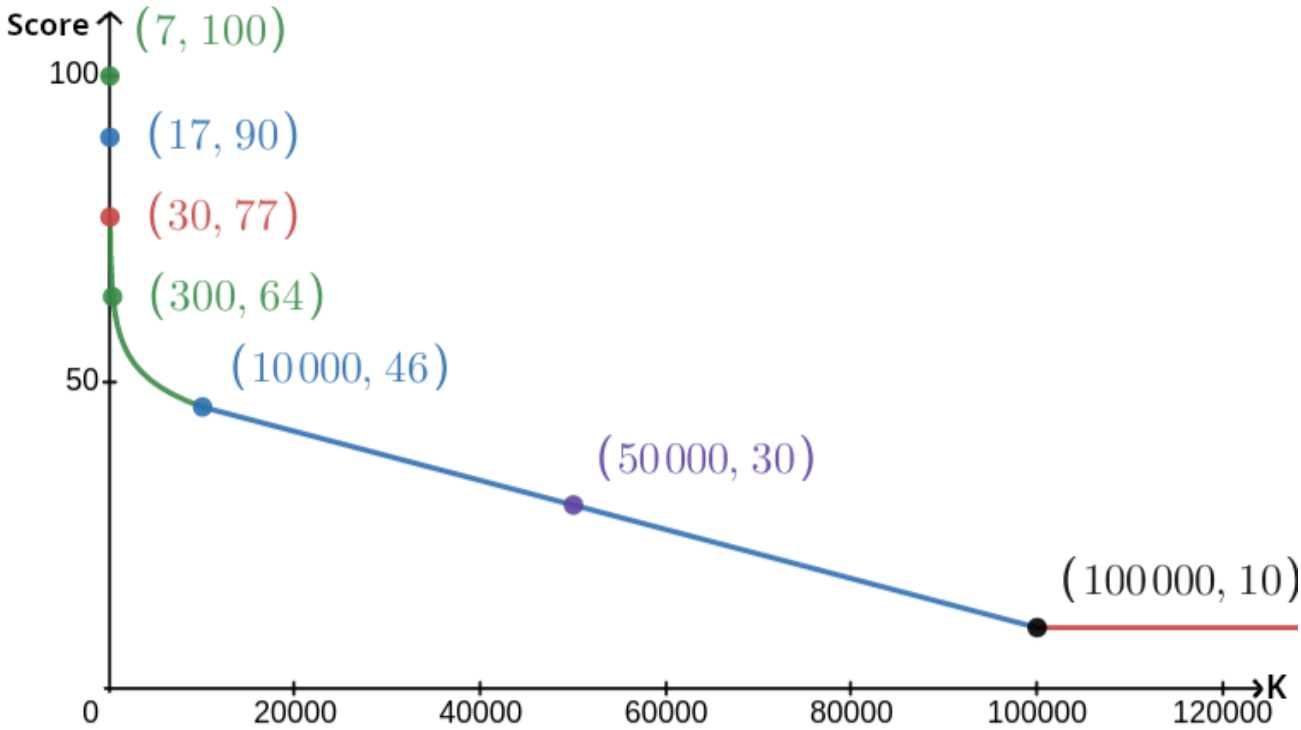
Punteggio

La tua soluzione verrà testata su una serie di casi di test. Se la tua soluzione fallisce su *uno qualsiasi* di questi casi di test (ad es. fornendo risposte errate ("Wrong Answer"), crashando ("Run Time Error"), superando il limite di tempo ("Time Limit Exceeded"), ecc.), riceverai 0 punti e il verdetto appropriato.

Se il tuo programma trova con successo l'indice segreto in *tutti* i casi di test, otterrai il verdetto "Accepted" e un punteggio calcolato come segue. Sia K_{max} il valore massimo di K utilizzato per qualsiasi test case. A seconda di K_{max}:

	Punteggio
$K_{max} > 99\,998$	10 punti
$10\,000 < K_{max} \leq 99\,998$	$10 + \lfloor 40(1 - \frac{K_{max}}{10^5}) \rfloor$ punti
$30 < K_{max} \leq 10\,000$	$46 + \lfloor 31(4 - \log_{10}(K_{max}))/(4 - \log_{10}(30)) \rfloor$ punti
$7 < K_{max} \leq 30$	$107 - K_{max}$ punti
$K_{max} \leq 7$	100 punti

La funzione di punteggio è illustrata nella figura seguente.



Il caso di test di esempio viene ignorato per il punteggio e non è necessario che la tua soluzione lo risolva.

Strumento di test

Per facilitare il test della tua soluzione, mettiamo a disposizione un semplice strumento che puoi scaricare. Vedi "allegati" in fondo alla pagina del problema di Kattis. L'uso dello strumento è facoltativo e puoi modificarlo. Si noti che il programma di valutazione ufficiale su Kattis è diverso dallo strumento di test.

Esempio di utilizzo (con $N = 4$, $s = 2$, dove s è il numero scritto sull'ultima casa):

Per i programmi Python, per esempio `solution.py` (normalmente eseguito come `pypy3 solution.py`):

```
python3 testing_tool.py pypy3 solution.py <<<"4 2"
```

Per i programmi C++, compila il programma normalmente (ad es. con `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`), dopodichè esegui:

```
python3 testing_tool.py ./solution.out <<<"4 2"
```

Lo strumento di test visiterà le case in ordine casuale. Per utilizzare un ordine specifico, modifica lo strumento di test nel punto in cui è scritto "MODIFY HERE".

Esempio di interazione

Il caso di test di esempio viene ignorato per il punteggio e la tua soluzione non deve funzionare su di esso.

Supponiamo di avere $N = 4$ e che Emma viva in casa 1. Sia A l'elenco dei numeri scritti sulle case. Inizialmente, $A = [0, 0, 0, 0]$, dove 0 significa che sulla casa corrispondente non è scritto alcun numero.

Nella prima esecuzione del tuo codice:

$N = 4$ è dato. La tua soluzione risponde con $K = 3$.

A_2 è richiesto. La tua soluzione risponde con 3. A ora è $[0, 0, 3, 0]$.

A_0 è richiesto. La tua soluzione risponde con 1. A ora è $[1, 0, 3, 0]$.

A_3 è richiesto. La tua soluzione risponde con 2. A ora è $[1, 0, 3, 2]$.

Infine, il grader imposta $A_1 = 2$, così che alla fine A = $[1, 2, 3, 2]$. Questo segna la fine della prima fase.

Nella Fase 2 dell'esecuzione del tuo codice, alla tua soluzione viene passata la lista `1 2 3 2`.

Risponde con `1 3`.

Poiché una delle risposte è l'indice corretto della casa (1), Anna e Bertil vincono la partita.

grader output	your output
1 4	
	3
2	
	3
0	
	1
3	
	2

grader output	your output
2 4	
1 2 3 2	
	1 3