D. Guessing Game

Nombre del problema	Guessing Game
Límite de tiempo	4 segundos
Límite de memoria	1 gigabyte

En el antiguo pueblo de Lund, hay una calle con N casas en fila, indexadas de 0 a $N-1$. Emma vive en una de estas casas y sus amigos Anna y Bertil quieren adivinar cuál. En vez de decirles a sus amigos dónde vive, Emma les propone un juego. Antes de que el juego comience, Anna y Bertil conocen el número de casas en la calle. Anna y Bertil pueden escoger un número K y ponerse de acuerdo en una estrategia. Cualquier comunicación posterior está prohibida.

El juego consiste de dos fases. En la primera fase, Emma escoge un orden en el que visitar las casas tal que la suya sea la última. Luego lleva a Anna a las casas en este orden, sin avisar a Anna previamente. Para cada casa que no sea la de Emma, Anna puede pintar un entero entre 1 y K en la puerta principal de cada casa con una tiza. Para la última casa, la de Emma, Emma escribe un número entre 1 y K.

En la segunda fase del juego, Bertil camina a través de la calle desde la casa 0 hasta la $N-1$ y lee todos los números escritos en las puertas por Anna y Emma. Ahora él querría saber en qué casa vive Emma. Tiene dos intentos para adivinar correctamente y Anna y él ganan si acierta. De lo contrario Emma gana el juego.

¿Puedes proponer una estrategia donde garantices que Anna y Bertil ganan el juego? Tu estrategia se evaluará en el valor de K (cuanto más pequeño, mejor).

Implementación

Este es un problema multi-ejecución, queriendo decir que tu programa se ejecutará múltiples veces. La primera vez que se ejecute, implementará la estrategia de Anna. Después implementará la estrategia de Bertil.

La primera línea de input contiene dos enteros, P y N, donde P es 1 o 2 (primera o segunda fase), y N es el número de casas. **Excepto en el input de ejemplo (que no se usa para la puntuación), N siempre será $100\,000$.**

El resto de input depende de la fase:

Fase 1

Tu programa debe empezar imprimiendo el número K en una línea $(1 \leq K \leq 1\,000\,000)$. Luego, $N-1$ veces, debe leer una línea que contiene un índice i $(0 \leq i < N)$, e imprimir una línea con un entero A_i $(1 \leq A_i \leq K)$, donde A_i es el número que Anna escribe en la puerta de la casa i. Cada índice i excepto el de la casa de Emma aparecerá una única vez, en un cierto orden decidido por el grader.

Fase 2

Tu programa debe leer una línea con N enteros, $A_0, A_1, \ldots, A_{N-1}$, donde A_i es el número escrito en la puerta de la casa i.

Luego, debe imprimir una línea con dos enteros, s_1 y s_2 $(0 \leq s_i < N)$, los posibles índices de la casa de Emma. s_1 y s_2 pueden ser iguales.

Detalles de la implementación

Ten en cuenta que cuando se ejecuta tu programa en la Fase 2, el programa se reinicia. Esto significa que no puedes guardar información en variables entre las fases.

Tras cada línea que imprimes, asegúrate de hacer flush, ya que si no lo haces tu programa podría recibir el veredicto Time Limit Exceeded. En Python, `print()` hace flush por defecto. En C++, `cout << endl;` también hace flush además de imprimir un salto de línea; si usas printf, usa `fflush(stdout)`.

El grader para este problema puede ser **adaptativo**, esto quiere decir que puede cambiar su comportamiento dependiendo de lo que haga tu programa para prevenir que soluciones heurísticas pasen como correctas. Puede ejecutar la Fase 1, mirar tu output, y luego ejecutar de nuevo la Fase 1 teniendo en cuenta la información que tiene sobre lo que hará tu programa.

Tu programa *debe* ser determinista, es decir, que siempre se comporte igual si se ejecuta varias veces con el mismo input. Si quieres que tu programa use funciones aleatorias, asegúrate de que use una seed fija. Esto se puede hacer pasándole una constante escrita a mano a la función `srand` (en C++) o `random.seed` (en Python), o, si usas generadores de números aleatorios de C++11, especificando la seed cuando construyes el generador de números aleatorios. En particular, no puedes usar `srand(time(NULL))` en C++. Si el grader detecta que tu programa no es determinista y está usando aleatoriedad, tu envío recibirá el veredicto Wrong Answer.

Si la *suma* de los tiempos de ejecución (de hasta 3) ejecuciones separadas de tu programa superan el límite de tiempo, tu envío recibirá el veredicto Time Limit Exceeded.

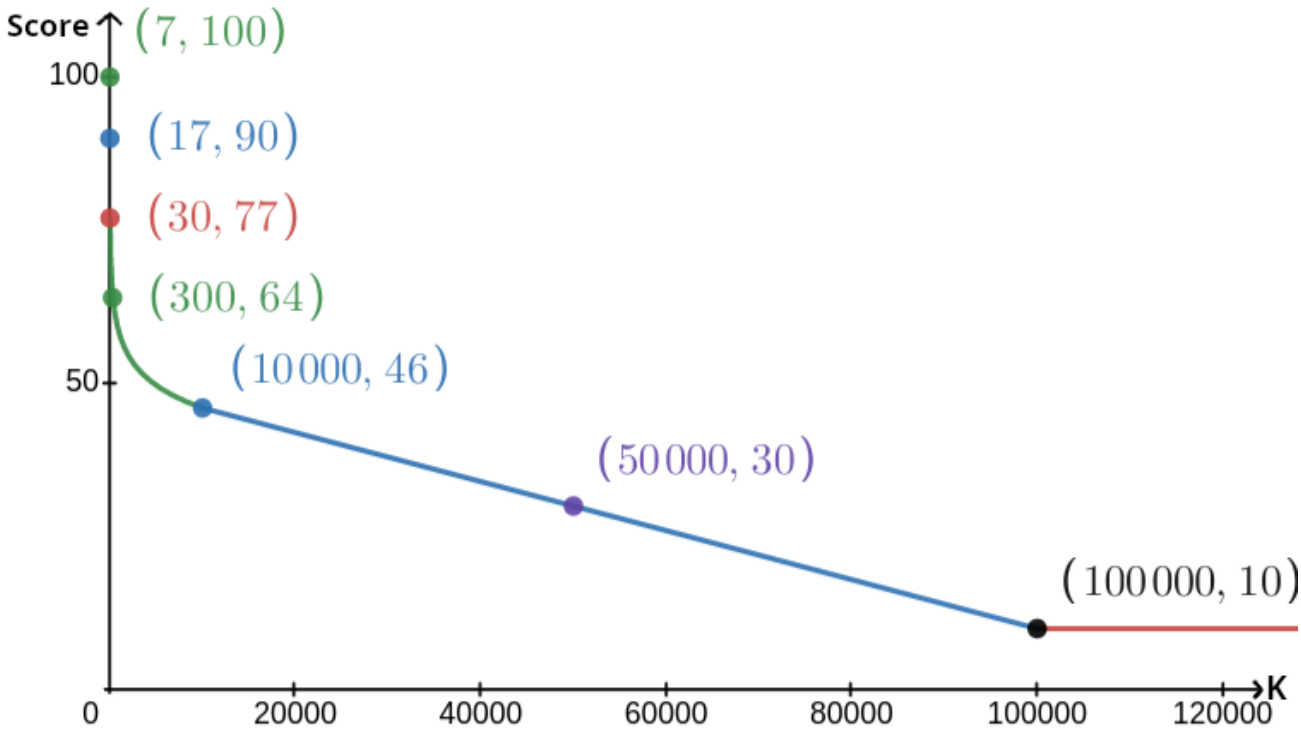
Puntuación

Tu solución será evaluada en un número de casos de prueba. Si tu solución falla *alguno* de estos casos (por ejemplo dando soluciones incorrectas (Wrong Answer), crashea (Run-Time Error), excediendo el límite de tiempo (Time Limit Exceeded), etc.), recibiras 0 puntos y el veredicto apropiado.

Si tu programa encuentra el índice de la casa de Emma correctamente en *todos* los casos, recibirás el veredicto Accepted, y la puntuación se calcula de la siguiente manera. Sea K_{max} el máximo valor de K utilizado en algún caso de prueba. Dependiendo de K_{max}:

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

La función de la puntuación se representa en la siguiente figura.



El caso de prueba es ignorado para el cálculo de la puntuación, y tu solución no tiene porque resolverlo.

Herramienta de testeo

Para facilitar el testeo de tu solución, te aportamos una herramienta simple que puedes descargar. Mira "attachments" al fondo de la página de problemas de Kattis. El uso de la herramienta es opcional y tienes permitido hacerle cambios. El grader oficial de Kattis es distinto de la herramienta de testeo.

Ejemplo de uso (con $N = 4$, $s = 2$, donde s es el número escrito en la última casa visitada):

Para los programas en Python, por ejemplo `solution.py` (normalmente ejecutados como `pypy3 solution.py`):

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

Para programas en C++, primero compílalos (por ejemplo con `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) y luego ejecuta:

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

La herramienta de testeo visitará las casas en un orden aleatorio. Para usar un orden específico, modifica la herramienta de testeo donde pone “MODIFY HERE”.

Interacción de ejemplo

El caso de prueba se ignora para el cálculo de la puntuación y tu solución no tiene porque funcionar en él.

Supongamos que tenemos $N = 4$ y que Emma vive en la casa 1. Llamemos A a la lista de los números escritos en las casas. Inicialmente, $A = [0, 0, 0, 0]$, donde 0 significa que no hay ningún número escrito en la casa correspondiente.

En la Fase 1 de tu código:

Se te da $N = 4$. Tu solución responde con $K = 3$.

Se te pregunta A_2. Tu solución responde con un 3. Ahora A es $[0, 0, 3, 0]$.

Se te pregunta A_0. Tu solución responde con un 1. Ahora A es $[1, 0, 3, 0]$.

Se te pregunta A_3. Tu solución responde con un 2. Ahora A es $[1, 0, 3, 2]$.

Finalmente, el grader elige $A_1 = 2$, por lo que $A = [1, 2, 3, 2]$ al final. Aquí acaba la primera fase.

En la Fase 2 de tu código, a tu solución se le da la lista `1 2 3 2`.

Tu solución responde con `1 3`.

Como uno de los dos posibles índices dados es el índice correcto de la casa (1), Anna y Bertil ganan el juego.

output del grader	tu output
1 4	
	3
2	
	3
0	
	1
3	
	2

output del grader	tu output
2 4	
1 2 3 2	
	1 3