D. Juego de adivinar

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

En la antigua ciudad 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 en cuál de ellas. En lugar de solo decirle a sus amigos en dónde vive, Emma ha decidido jugar un juego con ellos. Antes de iniciar el juego, Anna y Bertil solo conocen el número de casas en la calle. En este punto, Anna y Bertil pueden elegir un entero positivo K y acordar una estrategia. A partir de ese momento, toda comunicación entre ellos está prohibida.

El juego consiste de dos fases. En la primera fase, Emma elige un orden para visitar las casas de tal forma que su casa sea la última en ser visitada. Posteriormente lleva a Anna a las casas, una por una, en este orden, sin haber compartido antes el orden con Anna. Para cada casa diferente a la de Emma, Anna puede escribir un entero entre 1 y K en la puerta de la casa con un gis. Para la última casa que visitan, es decir, la casa de Emma, es Emma quien escribe un entero entre 1 y K en la puerta.

En la segunda fase del juego, Bertil camina a lo largo de la calle, de la casa 0 a la casa $N-1$, y lee todos los números que Anna y Emma escribieron en las puertas. Ahora él quiere adivinar cúal es la casa en la que vive Emma. Tiene dos oportunidades para adivinar correctamente, y Anna y él ganan el juego si lo logra. De lo contrario, Emma gana el juego.

¿Puedes diseñar una estrategia para garantizar que Anna y Bertil ganen el juego? Tu estrategia será evaluada con base en el valor de K (entre más pequeño, mejor).

Implementación

Tu programa será ejecutado en múltiples ocasiones. La primera vez que es ejecutado, implementa la estrategia de Anna. Posteriormente implementa la estrategia de Bertil.

La primera línea de entrada contiene dos enteros, P y N, donde P es 1 o 2 (primera o segunda fase) y N es el número de casas.

Con excepción de la entrada de ejemplo (que no se usa para la evaluación), N siempre es igual a

$$100\,000$$

La entrada que viene después de esto depende de la fase:

Fase 1

Tu programa debe comenzar imprimiendo el número K en una línea, $(1 \leq K \leq 1\,000\,000)$. A continuación, $N-1$ 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, con excepción del índice de la casa de Emma, aparecerá exactamente una vez en algún orden determinado por el evaluador.

Phase 2

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

Después, debe imprimir una línea con dos enteros s_1 y s_2 $(0 \leq s_i < N)$, los índices elegidos. s_1 y s_2 pueden ser iguales.

Detalles de implementación

Ten en cuenta que al ejecutar tu programa en la fase 2, tu programa se reinicia. Esto significa que no puedes guardar información en variables entre las ejecuciones.

Asegúrate de hacer flush del stream de salida después de cada línea que imprimas, de lo contrario tu programa podría obtener el veredicto Time Limit Exceeded. En Python, `print()` hace flush automáticamente. En C++, `cout << endl;` también hace flush además de imprimir una línea nueva; si usas `printf` deberás usar `fflush(stdout)`.

El evaluador para este problema puede ser **adaptativo**, esto significa que puede cambiar su comportamiento con base en la salida de tu programa para evitar que soluciones heuristicas obtengan puntos. El evaluador podría ejecutar la fase 1 como prueba, examinar tu salida, y luego ejecutar la fase 1 una vez más, como evaluación real, usando información obtenida de la primera ejecución.

Tu programa *debe* ser determinístico, es decir, comportarse igual si es ejecutado dos veces con la misma entrada. Si quieres introducir aleatoriedad en tu programa, asegúrate de usar un semilla aleatoria fija. Esto se puede lograr pasando una constante hard-codeada a `srand` (en C++), `random.seed` (en Python), o, si se usa el generador de números aleatorias de C++11, especificando la semilla al construir el generador de números aleatorios. En particular, no puedes

usar `srand(time(NULL))` en C++. Si el evaluador determina que tu programa no es determinístico, recibirás el veredicto Wrong Answer.

Si la *suma* de los tiempos de ejecución de las ejecuciones (a lo más 3) de tu programa excede el límite de tiempo, tu envío recibirá el veredicto Time Limit Exceeded.

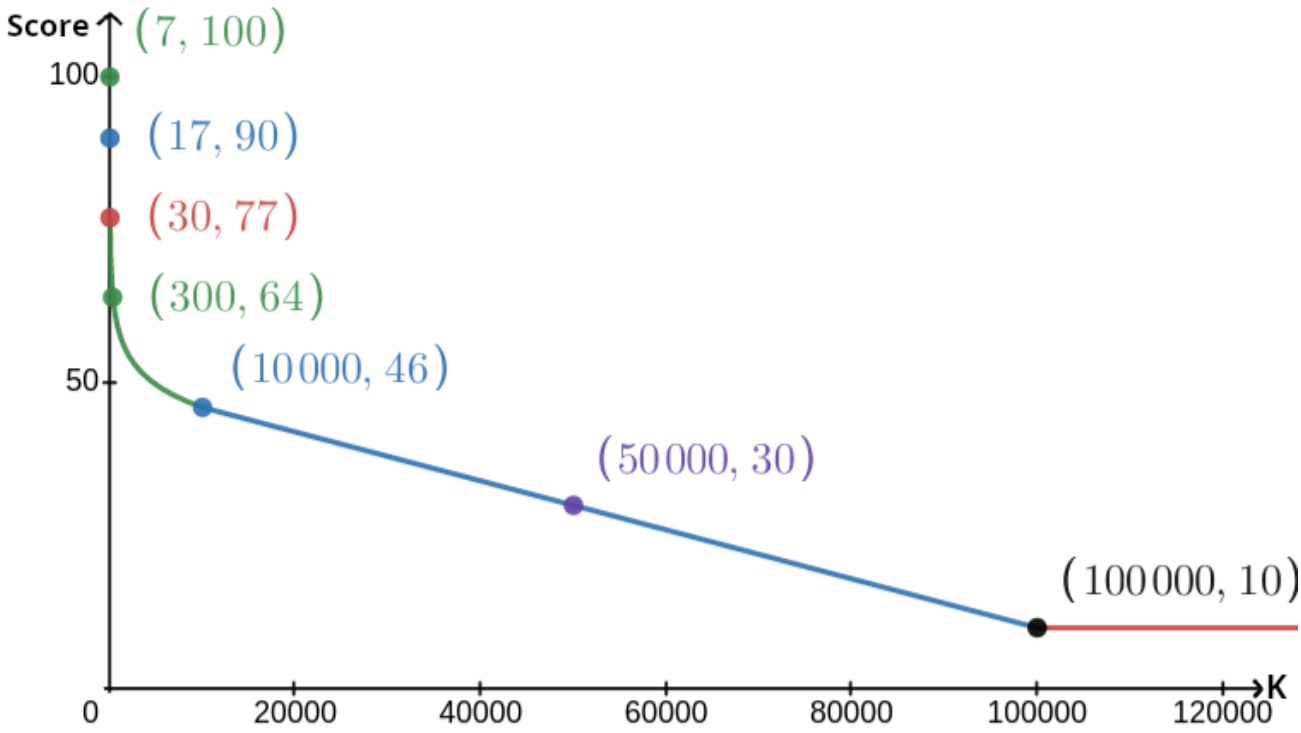
Evaluación

Tu solución se evaluará con un conjunto de grupos de casos de prueba. Si tu solución falla en *cualquiera* de estos casos de prueba (por ejemplo, si da una respuesta incorrecta (Wrong Answer), truena (Run-Time Error), o si excede el tiempo límite (Time Limit Exceeded), etc), recibirás 0 puntos y el veredicto correspondiente.

Si tu solución encuentra correctamente el índice de la casa de Emma en *todos* los casos de prueba, recibirás el veredicto Accepted, y un puntaje calculado como se describe a continuación. Sea K_{max} el máximo valor de K usado en algún caso de prueba. Dependiendo del valor de K_{max}:

	Puntaje
$K_{max} > 99\,998$	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 para determinar tu puntaje se muestra en la siguiente imagen.



El caso de prueba se ignora para la evaluación, por lo que no se requiere que tu solución funcione con este caso.

Herramienta de pruebas

Para facilitarte poder probar tu solución, te damos una herramienta simple que puedes descargar. Ve a "attachments" al final de la página del problema en Kattis. El uso de la herramienta es opcional y puedes cambiarla. Ten en cuenta que el evaluador oficial en Kattis es diferente a esta herramienta de pruebas.

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

Para programas en python, si tenemos solution.py (se corre normalmente pypy3 solution.py) corre:

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

For C++ programs, first compile it (e.g. with `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) and then run:

Para programas en C++, primero compílalo (por ejemplo g++ -std=gnu++17 solution.cpp -o solution.out) y luego corre:

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

La herramienta de pruebas visita las casas en orden aleatorio. Para usar un orden específico, modifica la herramienta de pruebas en la parte que dice “MODIFY HERE”.

Ejemplo de interacción

El caso de ejemplo es ignorado en la evaluación, y no se requiere que tu solución funcione con este caso.

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

En la primera ejecución de tu programa:

Tu solución recibe $N = 4$. Tu solución responde $K = 3$.

Se pide A_2. Tu solución responde 3. A es ahora $[0, 0, 3, 0]$.

Se pide A_0. Tu solución responde 1. A es ahora $[1, 0, 3, 0]$.

Se pide A_3. Tu solución responde 2. A es ahora $[1, 0, 3, 2]$.

Por último, el evaluador elige el valor de $A_1 = 2$, de tal modo que, al final, $A = [1, 2, 3, 2]$. Con esto se finaliza la fase 1.

En la fase 2 de tu programa, tu solución recibe la lista `1 2 3 2`.

Tu solución responde `1 3`.

Dado que uno de los dos valores elegidos coincide con el índice de la casa (1), Anna y Bertil ganan el juego.

Salida del evaluador	Tu salida
1 4	
	3
2	
	3
0	
	1
3	
	2

Salida del evaluador	Tu salida
2 4	
1 2 3 2	
	1 3