D. Jogo da Adivinhação

Nome do Problema	Jogo da Adivinhação
Limite de Tempo	4 segundos
Limite de Memória	1 gigabyte

Na antiga vila de Lund, há uma rua com N casas em sequência, indexadas de 0 a $N-1$. Emma mora em uma dessas casas e seus amigos Anna e Bertil querem descobrir qual. Ao invés de simplesmente dizer a seus amigos onde ela mora, Emma decide jogar um jogo com eles. Antes do início do jogo, Anna e Bertil sabem apenas a quantidade de casas na rua. Nesse momento, Anna e Bertil podem escolher um número inteiro positivo K e combinar uma estratégia. Qualquer comunicação depois disso é proibida.

O jogo em si consiste em duas fases. Na primeira fase, Emma escolhe uma uma ordem de visitação das casas, de modo que sua casa seja a última a ser visitada. Em seguida, ela leva Anna nas casas, uma a uma, nessa ordem, sem informar a ordem a Anna antes de começar. Para cada casa que não seja a casa de Emma, Anna pode escrever um único número inteiro entre 1 e K na porta da frente da casa com um pedaço de giz. Na última casa que visitarem, que é a casa da Emma, a própria Emma escreverá um número inteiro entre 1 e K na porta.

Na segunda fase do jogo, Bertil anda pela rua, indo da casa 0 até a casa $N-1$, e lê todos os números escritos nas portas por Anna e Emma. Agora ele quer adivinhar em qual casa Emma mora. Ele tem duas chances de adivinhar corretamente e ele e Anna ganham o jogo se ele conseguir. Caso contrário, a Emma vence o jogo.

Você consegue criar uma estratégia que garanta a vitória de Anna e Bertil no jogo? A sua estratégia receberá pontos com base no valor de K (quanto menor, melhor).

Implementação

Este é um problema comunicativo (em inglês, *multi-run*), o que quer dizer que o seu programa será executado várias vezes. Na primeira vez em que for executado, ele implementará a estratégia de Anna. Depois disso, ele implementará a estratégia de Bertil.

A primeira linha da entrada conterá dois números inteiros, P e N, onde P ou é 1 ou é 2 (primeira ou segunda fase) e N é a quantidade de casas. **Com exceção do exemplo de entrada (não usado**

para calcular a pontuação), N será sempre igual a $100\,000$.

A entrada a seguir depende da fase:

Fase 1

Seu programa deve começar imprimindo o número K em uma única linha $(1 \leq K \leq 1\,000\,000)$. Em seguida, $N-1$ vezes, ele deve ler uma linha contendo um índice i $(0 \leq i < N)$ e imprimir uma linha com um número inteiro A_i $(1 \leq A_i \leq K)$, onde A_i é o número que Anna escreve na porta da casa i. Cada índice i exceto o índice da casa da Emma aparecerá exatamente uma vez, em alguma ordem decidida pelo corretor.

Fase 2

Seu programa deve ler uma linha com N números inteiros, $A_0, A_1, \ldots, A_{N-1}$, onde A_i é o número escrito na porta da casa i.

Em seguida, ele deve imprimir uma linha com dois números inteiros, s_1 e s_2 $(0 \leq s_i < N)$, os índices adivinhados. s_1 e s_2 podem ser iguais.

Detalhes de Implementação

Note que ao executar seu programa na Fase 2, o programa é reiniciado. Isso significa que você não pode salvar as informações em variáveis entre as execuções.

Após cada linha imprimida, certifique-se de dar *flush* na saída padrão, caso contrário, seu programa poderá ser julgado como *Time Limit Exceeded*. Em Python, `print()` faz o *flush* automaticamente. Em C++, `cout << endl;` também faz o *flush* além de imprimir uma nova linha; se estiver usando printf, use `fflush(stdout)`.

O corretor para esse problema pode ser **adaptativo**, o que significa que ele pode mudar seu comportamento dependendo da saída do seu programa para evitar que soluções heurísticas sejam aceitas. Ele pode fazer uma execução de teste da fase 1, verificar sua saída e, em seguida, executar a fase 1 de verdade usando as informações extraídas da execução anterior.

Seu programa *precisa* ser determinístico, ou seja, comportar-se da mesma forma se for executado duas vezes com a mesma entrada. Se você quiser usar aleatoriedade em seu programa, certifique-se de usar uma semente aleatória fixa. Isso pode ser feito passando uma constante de forma *hard-coded* (ou seja, escolhendo um valor fixo para ela e colocando no código) para `srand` (em C++) ou `random.seed` (em Python) ou, se estiver usando os geradores de números aleatórios do C++11, especificando a semente ao construir o gerador de números aleatórios. Em particular, você não pode usar `srand(time(NULL))` em C++. Se o avaliador detectar que seu programa não é determinístico, ele receberá o veredito *Wrong Answer*.

Se a *soma* dos tempos de execução das (até 3) execuções separadas do seu programa exceder o limite de tempo, sua submissão será julgada como *Time Limit Exceeded*.

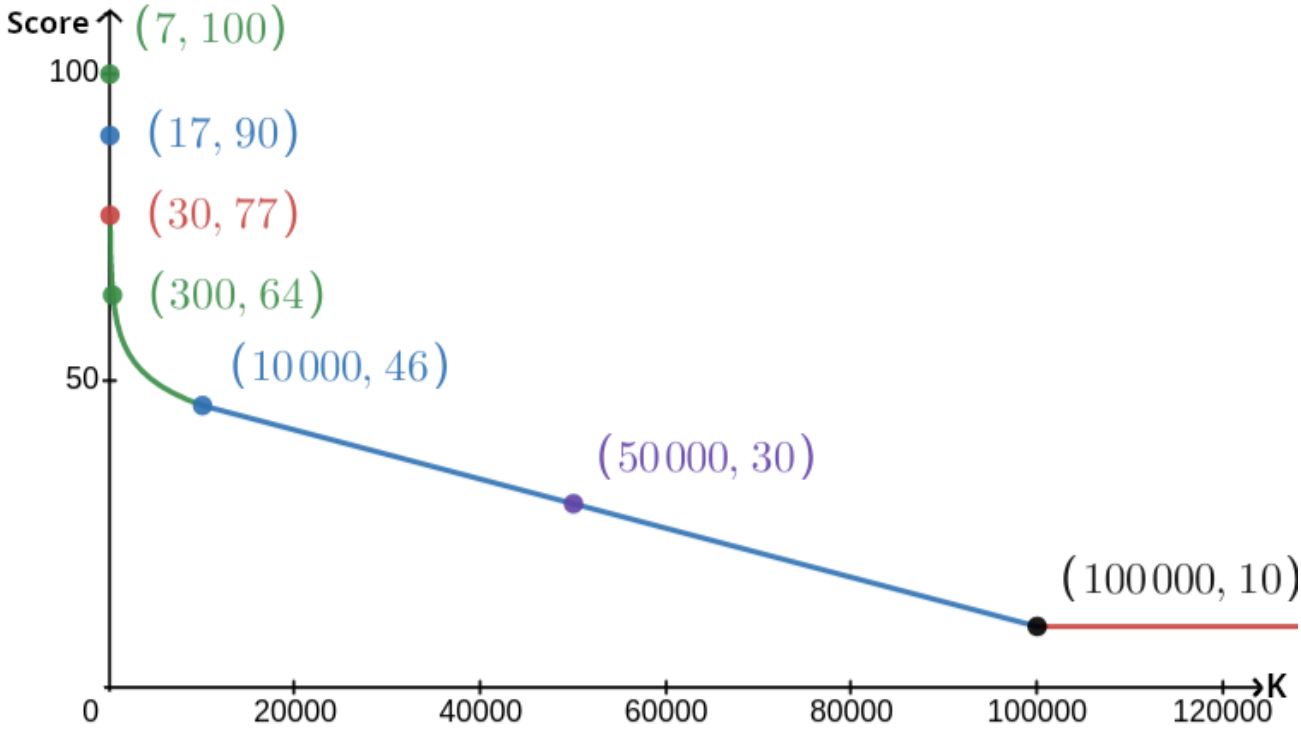
Pontuação

Sua solução será testada em vários casos de teste. Se a sua solução falhar em *qualquer* desses casos de teste (por exemplo, dando respostas erradas (*Wrong Answer*), encerrando abruptamente (*Run-Time Error*), excedendo o limite de tempo (*Time Limit Exceeded*), etc.), você receberá 0 pontos e o veredito apropriado.

Se o seu programa encontrar com sucesso o índice da casa da Emma em *todos* os casos de teste, você receberá o veredito *Accepted* e uma pontuação calculada da maneira a seguir. Seja K_{max} o valor máximo de K usado em qualquer caso de teste. Dependendo de K_{max}:

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

A função da pontuação está ilustrada na figura abaixo.



O exemplo de caso de teste é ignorado para a pontuação e sua solução não precisa funcionar para ele.

Corretor Exemplo

Para facilitar o teste de sua solução, fornecemos um corretor exemplo simples que você pode baixar. Consulte "*attachments*" (anexos) na parte inferior da página do problema no Kattis. O uso do corretor exemplo é opcional e você tem permissão para alterá-lo. Note que o corretor oficial no Kattis é diferente do corretor exemplo.

Exemplo de uso (com $N = 4$, $s = 2$, onde s é o número escrito na última casa visitada):

Para programas em Python, por exemplo, `solution.py` (normalmente executado como `pypy3 solution.py`):

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

Para programas em C++, primeiro compile-o (por exemplo, com `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) e, em seguida, execute:

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

O corretor exemplo visitará as casas em ordem aleatória. Para usar uma ordem específica, modifique o corretor exemplo onde está escrito "MODIFY HERE".

Exemplo de Interação

O exemplo de caso de teste é ignorado para pontuação e sua solução não precisa funcionar para ele.

Suponha que temos $N = 4$ e que a Emma mora na casa 1. Seja A a lista de números escritos nas casas. Inicialmente, $A = [0, 0, 0, 0]$, onde 0 significa que nenhum número está escrito na casa correspondente.

Na primeira execução de seu código:

É dado $N = 4$. Sua solução responde com $K = 3$.

É solicitado A_2. Sua solução responde com 3. Agora A é $[0, 0, 3, 0]$.

É solicitado A_0. Sua solução responde com 1. Agora A é $[1, 0, 3, 0]$.

A_3 é solicitado. Sua solução responde com 2. Agora A é $[1, 0, 3, 2]$.

Por fim, o corretor define $A_1 = 2$, de modo que A = $[1, 2, 3, 2]$ no final. Isso marca o fim da primeira fase.

Na Fase 2 do seu código, sua solução recebe a lista `1 2 3 2`.

A sua solução responde com `1 3`.

Como um dos índices adivinhados é o índice correto da casa (1), Anna e Bertil vencem o jogo.

saída do corretor	sua saída
1 4	
	3
2	
	3
0	
	1
3	
	2

saída do corretor	sua saída
2 4	
1 2 3 2	
	1 3