D. Guessing Game

Problemname	Guessing Game
Time Limit	4 Sekunden
Memory Limit	1 Gigabyte

In Lunds Altstadt gibt es eine Strasse mit N Häusern in einer Reihe, die von 0 bis $N-1$ indiziert sind. Emma lebt in einem dieser Häuser und ihre Freunde Anna und Bertil wollen herausfinden in welchem. Anstatt ihnen direkt zu sagen, in welchem Haus sie wohnt, entscheidet sich Emma, ein Spiel mit ihnen zu spielen. Bevor das Spiel beginnt, kennen Anna und Bertil nur die Anzahl der Häuser in der Strasse. Zu diesem Zeitpunkt können Anna und Bertil eine positive ganze Zahl K wählen und sich auf eine Strategie einigen. Danach ist jegliche Kommunikation verboten.

Das Spiel selbst besteht aus zwei Phasen. In der ersten Phase wählt Emma eine Reihenfolge, in der die Häuser besucht werden sollen, so, dass ihr Haus das letzte ist, welches besucht wird. Dann führt sie Anna in genau dieser Reihenfolge zu den Häusern, ohne Anna diese Reihenfolge vorher mitzuteilen. Für jedes Haus, das nicht Emmas Haus ist, darf Anna eine einzelne, ganze Zahl zwischen 1 und K mit Kreide an die Haustür schreiben. Beim letzten Haus, Emmas Haus, schreibt Emma selbst eine beliebige ganze Zahl zwischen 1 und K an die Tür.

In der zweiten Phase des Spiels läuft Bertil die Strasse von Haus 0 bis Haus $N-1$ entlang und liest die Nummern, die von Anna und Emma auf die Türen geschrieben wurden. Dann möchte er raten, in welchem Haus Emma wohnt. Dazu darf er genau zweimal raten und er und Anna gewinnen genau dann, wenn er dabei Emmas Haus errät. Ansonsten gewinnt Emma.

Kannst du eine Strategie entwickeln, mit der Anna und Bertil definitiv gewinnen werden? Deine Strategie wird anhand des Werts K bepunktet (je kleiner, desto besser).

Implementation

Das ist eine multi-run-Aufgabe, was bedeutet, dass dein Programm mehrfach ausgeführt wird. Bei der ersten Ausführung wird es Annas Strategie implementieren, bei der zweiten Ausführung Bertils.

Die erste Zeile der Eingabe enthält zwei ganze Zahlen P und N, wobei P entweder 1 oder 2 (erste oder zweite Phase) und N die Anzahl der Häuser ist. **Ausser bei der Beispieleingabe (welches**

nicht für die Bepunktung genutzt wird) ist N immer $100\,000$.

Die danach folgende Eingabe hängt von der aktuellen Phase ab:

Phase 1

Dein Programm soll damit anfangen, die Zahl K in einer einzelnen Zeile auszugeben $(1 \leq K \leq 1\,000\,000)$. Dann soll es $N-1$-mal eine Zeile einlesen, die einen Index i $(0 \leq i < N)$ enthält, und eine Zeile ausgeben, die eine ganze Zahl A_i $(1 \leq A_i \leq K)$ enthält, wobei A_i die Nummer ist, die Anna auf die Tür i schreibt. Jeder Index i ausser dem Index von Emmas Haus taucht genau einmal in einer zufälligen Reihenfolge, die der Grader entscheidet, auf.

Phase 2

Dein Programm soll eine Zeile mit N ganzen Zahlen, $A_0, A_1, \ldots, A_{N-1}$, einlesen, wobei A_i die Zahl ist, die an der Tür von Haus i steht.

Dann soll es mit zwei ganzen Zahlen, s_1 und s_2 $(0 \leq s_i < N)$, ausgeben - den geratenen Indizes. s_1 und s_2 dürfen identisch sein.

Implementierungsdetails

Beachte, dass, wenn dein Programm in Phase 2 läuft, es vorher neu gestartet wird. Das bedeutet, dass du keine Informationen in Variablen zwischen zwei Läufen speichern kannst.

Nach jeder Zeile, die du ausgibst, achte darauf die Standarteingabe zu flushen. Ansonsten wird dein Programm gegebenenfalls mit Time Limit Exceeded bewertet. In Python flusht `print()` automatisch. In C++ flusht `cout << endl;` neben dem Ausgeben einer neuen Zeile; wenn du printf benutzt, nutze `fflush(stdout)`.

Der Grader von dieser Aufgabe ist **adaptiv**, was bedeutet, dass sich sein Verhalten je nach Ausgabe deines Programms ändert, um zu verhindern, dass heuristische Lösungen akzeptiert werden. Der Grader könnte einen Probelauf für Phase 1 ausführen, deinen Output betrachten und dann Phase 1 mit den neu erhaltenen Informationen vom vorherigen Lauf wirklich ausführen.

Dein Programm *muss* deterministisch sein, damit es sich bei zwei Einführungen auf dem selben Input gleich verhält. Wenn du Zufall in deinem Programm benutzen möchtest, dann stelle sicher, dass du es mit einem festgelegten Random-Seed tust. Dies kannst du machen, indem du eine hartgecodete Konstante in die Funktion `srand` (in C++) oder `random.seed` (in Python) übergibst oder, sofern du C++ 11s Zufallszahlengenerator benutzt, indem du den Seed beim Erstellen des Zufallszahlengenerators spezifizierst. Insbesondere kannst du nicht `srand(time(NULL))` in C++ benutzen. Wenn der Grader feststellt, dass dein Programm nicht deterministisch ist, wirst du das Feedback Wrong Answer bekommen.

Wenn die *Summe* der Laufzeiten von den (bis zu 3) separaten Ausführungen deines Programms die erlaubt Laufzeit überschreitet, wird deine Submission mit Time Limit Exceeded bewertet.

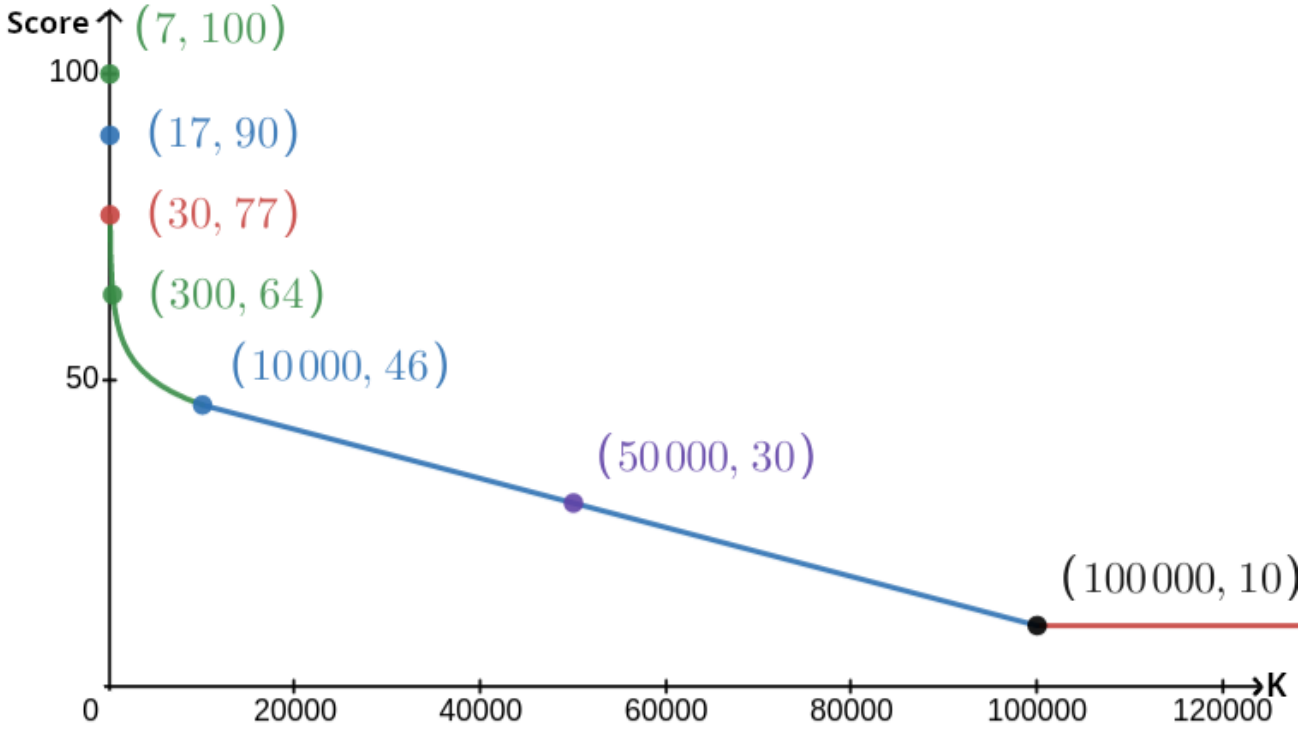
Bewertung

Deine Lösung wird auf mehreren Testfällen getestet. Wenn deine Lösung auf *einem* Testfall fehlschlägt (z.B. indem es eine falsche Antwort gibt (Wrong Answer), crasht (Run-Time Error), das Timelimit überschreitet (Time Limit Exceeded), etc.), bekommst du 0 Punkte und die entsprechende Bewertung.

Wenn dein Programm den Index von Emmas Haus in *allen* Testfällen erfolgreich findet, bekommst du die Bewertung Accepted, und deine Punktzahl wird wie folgend errechnet. Sei K_{max} das Maximum aller K_{max}s die du in einem Testfall genutzt hast. In Abhängigkeit von K_{max}:

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

Die Bewertungsfunktion ist im Folgenden dargestellt.



Der Beispieltestfall wird beim Bewerten ignoriert, und deine Lösung muss auf diesem nicht funktionieren.

Test-Tool

Um das Testen deiner Lösung zu vereinfachen, stellen wir dir ein einfaches Tool zur Verfügung, das du downloaden kannst. Siehe "attachments" unten auf der Kattis Aufgabenseite. Das Tool kann man freiwillig nutzen und du darfst es ändern. Beachte, dass das offizielle Graderprogramm auf Kattis anders ist als das Test-Tool.

Beispielnutzung (mit $N = 4$, $s = 2$, wobei s die Zahl ist, die auf das letzte besuchte Haus geschrieben wurde):

Für Python Programme, rufe `solution.py` auf (normalerweise ausgeführt als `pypy3 solution.py`):

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

Für C++ Programme, kompiliere es zuerst (z.B. mit `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) und führe dann Folgendes aus:

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

Das Test-Tool wird alle Häuser in einer zufälligen Reihenfolge besuchen. Um eine spezifische Ordnung zu nutzen, musst du das Test-Tool dort abändern, wo es “MODIFY HERE” heisst.

Beispielinteraktion

Der Beispieltestfall wird beim Bewerten ignoriert, und deine Lösung muss auf diesem nicht funktionieren.

Angenommen $N = 4$ und Emma lebt im Haus 1. Sei A die Liste an Zahlen die an die Häuser geschrieben wurde. Am Anfang ist $A = [0, 0, 0, 0]$, wobei 0 meint, dass keine Zahl auf dem dazugehörigen Haus steht.

Erste Ausführung deines Codes:

$N = 4$ ist gegeben. Deine Lösung antwortet mit $K = 3$.

A_2 wird abgefragt. Deine Lösung antwortet mit 3. A ist jetzt $[0, 0, 3, 0]$.

A_0 wird abgefragt. Deine Lösung antwortet mit 1. A ist jetzt $[1, 0, 3, 0]$.

A_3 wird abgefragt. Deine Lösung antwortet mit 2. A ist jetzt $[1, 0, 3, 2]$.

Schlussendlich setzt der Grader $A_1 = 2$, sodass am Ende $A = [1, 2, 3, 2]$ ist. Dies markiert das Ende der ersten Phase.

In Phase 2 deines Codes, wird deiner Lösung `1 2 3 2` übergeben.

Er antwortet mit `1 3`.

Da einer der beiden geratenen Indexe der des Hauses (1) ist, gewinnen Anna und Bertil das Spiel.

Graderausgabe	deine Ausgabe
1 4	
	3
2	
	3
0	
	1
3	
	2

Graderausgabe	deine Ausgabe
2 4	
1 2 3 2	
	1 3