D. Guessing Game

Numele problemei	Guessing Game
Limită de timp	4 seconds
Limită de memorie	1 gigabyte

În partea veche a orașului Lund, există o stradă cu N case în rând, indexate de la 0 la $N-1$. Emma locuiește într-una dintre aceste case și prietenii ei, Anna și Bertil, doresc să afle în care dintre ele. În loc să le spună pur și simplu unde locuiește, Emma decide să joace un joc cu ei. Înainte de a începe jocul, Anna și Bertil știu doar numărul de case de pe stradă. În acest moment, Anna și Bertil pot alege un număr întreg pozitiv K și pot conveni asupra unei strategii. Orice comunicare ulterioară este interzisă.

Jocul în sine constă din două faze. În prima fază, Emma alege o ordine în care să viziteze casele, astfel încât casa ei să fie ultima vizitată. Apoi o conduce pe Anna la case, una câte una, în această ordine, fără să-i spună ordinea din start. Pentru fiecare casă care nu este casa Emmei, Anna are voie să scrie pe ușa din față a casei cu o bucată de cretă un singur număr întreg între 1 și K. Pe ultima casă pe care o vizitează, casa Emmei, Emma scrie însăși pe ușă un număr întreg între 1 și K.

În a doua fază a jocului, Bertil merge pe stradă de la casa 0 la casa $N-1$ și citește toate numerele scrise de Anna și Emma pe uși. Acum el vrea să ghicească în care casă locuiește Emma. Are două șanse să ghicească corect. Dacă Bertil ghicește atunci el și Anna câștigă jocul, în caz contrar, Emma câștigă jocul.

Puteți concepe o strategie în care Anna și Bertil garantat câștigă jocul? Strategia propusă va fi punctată pe baza valorii K (cu cât este mai mică, cu atât mai bine).

Implementarea

Aceasta este o problemă cu mai multe rulări, ceea ce înseamnă că programul dvs. va fi executat de mai multe ori. La prima rulare va implementa strategia Annei. Apoi va implementa strategia lui Bertil.

Prima linie din input va conține două numere întregi, P și N, unde P este sau 1, sau 2 (prima sau a doua fază), iar N este numărul de case. **Cu excepția exemplului de test propus (care nu este**

folosit pentru punctare), N **va fi întotdeauna egal cu** $100\,000$.

Următorul input depinde de fază:

Faza 1

Programul ar trebui să înceapă prin afișarea numărului K pe o singură linie $(1 \leq K \leq 1\,000\,000)$. Apoi, de $N-1$ ori, va trebui să citească o linie care conține un index i $(0 \leq i < N)$, și să afișeze o linie cu un număr întreg A_i $(1 \leq A_i \leq K)$, unde A_i este numărul scris de Anna pe ușa casei i. Fiecare index i cu excepția indexului oferit casei Emelei, va apărea exact o dată, într-o anumită ordine decisă de grader.

Faza 2

Programul ar trebui să citească o linie cu N numere întregi, $A_0, A_1, \ldots, A_{N-1}$, unde A_i este numărul scris pe ușa casei i.

Apoi, ar trebui să afișeze o linie cu două numere întregi, s_1 și s_2 $(0 \leq s_i < N)$, care sunt indicii ghiciți. s_1 și s_2 pot fi egali.

Detalii de implementare

Observați că atunci când rulați programul în Faza 2, programul este repornit. Aceasta înseamnă că nu puteți salva informațiile în variabile între rulări.

După fiecare afișare dați flush la output, altfel programul poate lua verdictul Time Limit Exceeded. În Python, `print()` dă flush automat. În C++, `cout << endl;` dă flush în adiția printării unei noi linii; în cazul folosirii funcției printf, folosește `fflush(stdout)`.

Graderul pentru această problemă poate fi **adaptiv**, ceea ce înseamnă că își poate schimba comportamentul în funcție de ieșirea programului dvs. pentru a preveni trecerea soluțiilor euristice. S-ar putea să facă o execuție de probă a fazei 1, să examineze ieșirea programului și apoi să ruleze faza 1 în mod real, folosind informațiile extrase din rularea anterioară.

Programul *trebuie* să fie determinist, adică să se comporte la fel dacă rulează pe aceleași date de intrare. Dacă doriți să utilizați randomizare în program, asigurați-vă că utilizați o stare de pornire (seed) aleatorie fixă.

Acest lucru se poate face prin transmiterea unei constante hard-coded către `srand` (în C++) sau `random.seed` (în Python), sau dacă folosiți generatorul de numere aleatoare în C++11 prin specificarea stării de pornire (seed) la construirea generatorului de numere aleatoare. În particular, nu puteți folosi `srand(time(NULL))` în C++. Dacă graderul detectează că programul nu este determinist, i se va da verdictul Wrong Answer (Răspuns Greșit).

Dacă *suma* timpilor de execuție (până la 3) a rulărilor separte ale programului depășește limita de timp, soluția va fi considerată ca Time Limit Exceeded (Limită de timp depășită).

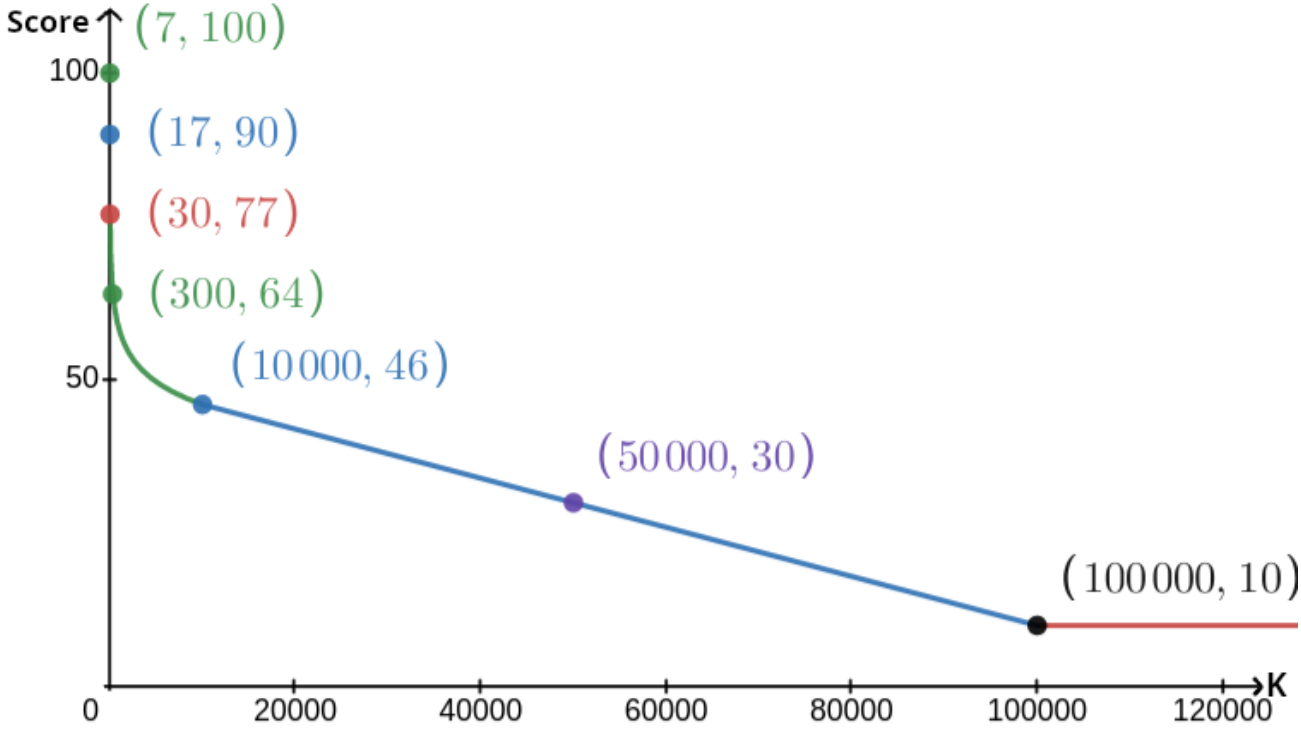
Punctaj

Soluția va fi testată pe mai multe grupe de teste. Dacă soluția eșuează în *oricare* dintre aceste grupe (de exemplu, prin raportarea unei poziții greșite a cutiei (Wrong answer), prin erori de rulare (Runtime Error), depășind limita de timp Time Limit Exceeded), etc.), vei primi 0 puncte și verdictul corespunzător

Dacă programul găsește cu succes indicele casei Emelei în *toate* grupele de test, veți primi verdictul Accepted (Acceptat) și un punctaj calculat după cum urmează. Fie K_{max} valoarea maximă a lui K utilizată pentru orice grup de teste. În funcție de K_{max}:

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

Funcția de punctare este prezentată în figura de mai jos.



Exemplul de test prezentat este ignorat pentru punctare și soluția nu trebuie să funcționeze pentru el.

Instrument de testare

Pentru a facilita testarea soluției dvs., oferim un instrument simplu pe care îl puteți descărca. Consultați secțiunea "attachments" din partea de jos a paginii Kattis a problemei. Utilizarea instrumentului este opțională, și aveți permisiunea să îl modificați. Rețineți că graderul oficial de pe Kattis este diferit de instrumentul de testare.

Exemplu de utilizare (cu $N = 4$, $s = 2$, unde s este numărul scris pe ultima casă vizitată):

Pentru programele Python, `solution.py` (rulează normal cu `pypy3 solution.py`):

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

Pentru programele C++, mai întâi compilați-l (de exemplu cu `g++ -g -O2 -std=gnu++17 -static solution.cpp -o solution.out`) și apoi rulați-l:

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

Instrumentul de testare va vizita casele în ordine aleatorie. Pentru a utiliza o anumită ordine, modificați instrumentul de testare unde scrie “MODIFY HERE”.

Exemplu de interacțiune

Exemplul de test prezentat este ignorat pentru punctare, iar soluția dvs. nu trebuie să funcționeze pentru el.

Să presupunem că avem $N = 4$ și că Emma locuiește în casă 1. Fie A lista de numere scrise pe case. Inițial, $A = [0, 0, 0, 0]$, unde 0 înseamnă că nu este scris niciun număr pe casa corespunzătoare.

La prima rulare a codului dvs.:

$N = 4$ este dat. Soluția dvs. răspunde cu $K = 3$.

Se solicită A_2. Soluția dvs. răspunde cu 3. A acum este $[0, 0, 3, 0]$.

Se solicită A_0. Soluția dvs. răspunde cu 1. A acum este $[1, 0, 3, 0]$.

Se solicită A_3. Soluția dvs. răspunde cu 2. A acum este $[1, 0, 3, 2]$.

În cele din urmă, graderul setează $A_1 = 2$, astfel încât $A = [1, 2, 3, 2]$. Aceasta marchează sfârșitul primei faze.

În Faza 2 a codului, soluției dvs. i se transmite lista `1 2 3 2`.

Acesta returnează `1 3`.

Deoarece una dintre presupuneri este indexul corect al casei (1), Anna și Bertil câștigă jocul.

Ieșirea grader	Ieșirea dvs
1 4	
	3
2	
	3
0	
	1
3	
	2

Ieșirea grader	Ieșirea dvs
2 4	
1 2 3 2	
	1 3