Zadanie: CYK
Przesunięcie cykliczne

Olimpiada Informatyczna

XXVI OI, etap II, dzień drugi. Plik źródłowy `cyk.*` Dostępna pamięć: 64 MB. *14.02.2019*

Opiekowanie się małym dzieckiem jest trudną pracą, szczególnie jeśli to dziecko uwielbia zabawy z ciągami liczb. Bajtalina ma pewien ciąg rosnący złożony z n liczb całkowitych, $a_1 < a_2 < \ldots < a_n$. Zostałeś zaproszony do zabawy polegającej na zadawaniu pytań, których celem jest odkrycie długości ciągu (czyli liczby n). Każde pytanie jest następującej postaci:

Podajesz Bajtalinie liczbę całkowitą x. Dziewczynka przesuwa ciąg cyklicznie o x elementów, czyli x-krotnie przenosi pierwszy element na koniec ciągu. Po takiej operacji Bajtalina podaje Ci wartość pierwszego elementu ciągu.

By jeszcze bardziej utrudnić Twoje zadanie, tuż przed zabawą Bajtalina sama wybiera pewną wartość x_0 i przesuwa ciąg cyklicznie o x_0 elementów. Oznacza to, że początkowy ciąg niekoniecznie jest rosnący, ale na pewno istnieje jego przesunięcie cykliczne, które jest rosnące.

Czy jesteś w stanie znaleźć długość ciągu, zadając co najwyżej 100 pytań?

Komunikacja

Twój program powinien używać biblioteki, która pozwala na zadawanie pytań Bajtalinie oraz na udzielenie ostatecznej odpowiedzi. Aby użyć biblioteki, należy wpisać na początku programu:

- **C++:** `#include "cyklib.hpp"`
- **Python:** `from cyklib import circular_shift, give_answer`

Biblioteka udostępnia następujące dwie funkcje:

- `circular_shift`(x)
  Jest to funkcja, której należy użyć do zadania pytania Bajtalinie. Przekazana wartość x ($0 \leq x \leq 10^9$) jest liczbą przeniesień pierwszego elementu ciągu na koniec, które wykona Bajtalina. Funkcja zwraca wartość pierwszego elementu ciągu po tej operacji.
  - **C++:** `long long circular_shift(int x);`
  - **Python:** `def circular_shift(x)`
- `give_answer`(n)
  Jest to funkcja, którą należy wywołać dokładnie raz, na koniec działania Twojego programu. Odpowiada ona Bajtalinie, że n to odgadnięta długość ciągu.
  - **C++:** `void give_answer(int n);`
  - **Python:** `def give_answer(n)`

Twój program **nie może** czytać żadnych danych (ani ze standardowego wejścia, ani z plików). **Nie może** również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (`stderr`) – pamiętaj jednak, że zużywa to cenny czas.

Po użyciu funkcji `give_answer` Twój program powinien natychmiastowo zakończyć działanie.

Przykładowy przebieg programu

Rozważmy ciąg rosnący $(0, 5, 15, 17, 18)$, który Bajtalina przesunęła na początku cyklicznie o $x_0 = 2$. Początkowy ciąg to zatem $(15, 17, 18, 0, 5)$.

Wywołanie	Ciąg (a_i) po wywołaniu	Zwrócona wartość
`circular_shift(2)`	$(18, 0, 5, 15, 17)$	18
`circular_shift(1)`	$(0, 5, 15, 17, 18)$	0
`circular_shift(8)`	$(17, 18, 0, 5, 15)$	17
`circular_shift(0)`	$(17, 18, 0, 5, 15)$	17
`circular_shift(8)`	$(5, 15, 17, 18, 0)$	5
`circular_shift(1)`	$(15, 17, 17, 0, 5)$	15
`give_answer(5)`	–	–

Przed pierwszym zapytaniem ciąg to $(15, 17, 18, 0, 5)$. Operacja `circular_shift(2)` powoduje dwukrotne przeniesienie pierwszego elementu na koniec ciągu. Po pierwszym takim przeniesieniu mamy ciąg $(17, 18, 0, 5, 15)$, a po drugim $(18, 0, 5, 15, 17)$. Wywołana funkcja zwróci wartość pierwszego elementu nowego ciągu czyli 18.

Ocenianie

We wszystkich testach ukryty ciąg Bajtaliny spełnia następujące warunki:

- $1 \leq n \leq 500\,000$;
- $0 \leq a_i \leq 10^{18}$ dla każdego $1 \leq i \leq n$.

W każdym teście ciąg (a_i) jest ustalony i nie zmieni się pomiędzy zapytaniami. Oznacza to, że np. dla powyższego testu przykładowego wywołanie funkcji `give_answer(5)` i zakończenie działania programu bez jakichkolwiek zapytań jest dopuszczalne i zaliczyłoby ten test. Możesz więc próbować zgadywać długość ciągu bez bycia pewnym odpowiedzi.

Zestaw testów dzieli się na następujące podzadania z dodatkowymi warunkami. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	**Warunki**	**Liczba punktów**
1	$n \leq 10$	9
2	$n \leq 2000$	33
3	$x_0 = 0$, czyli początkowy ciąg (a_i) jest rosnący	22
4	brak dodatkowych warunków	36

Eksperymenty

W katalogu `dlazaw` znajdują się pliki, które pozwolą Ci przetestować poprawność formalną rozwiązania. Możesz tam znaleźć następujące pliki:

- **C++:** bibliotekę `cyklib.hpp` i przykładowy błędny program `cyk.cpp`
- **Python:** bibliotekę `cyklib.py` i przykładowy błędny program `cyk.py`

Zwróć uwagę, że biblioteki te różnią się od tych, przy pomocy których będzie finalnie oceniane Twoje rozwiązanie, i służą jedynie do sprawdzenia poprawnej interakcji. Znajdując się w katalogu `dlazaw`, możesz standardowo skompilować i uruchomić `cyk.cpp` lub `cyk.py`. Przykładowa kompilacja w języku C++ to:

- `g++ -O3 -static cyk.cpp -std=c++11 -o cyk.e`

Tak otrzymany program wczytuje z wejścia liczbę n i początkowe przesunięcie x_0 (w pierwszym wierszu) oraz rosnący ciąg (a_i) przed przesunięciem o x_0 (w drugim wierszu). Format ten jest stosowany w plikach „ocen” w folderze `in`.

Pamiętaj, że dostępna przykładowa biblioteka nie sprawdza, czy dane na wejściu są sformatowane poprawnie ani czy spełnione są ograniczenia z treści zadania. Aby uruchomić swój program na pierwszym z testów „ocen”, użyj jednego z następujących poleceń:

- **C++:** `./cyk.e < in/cyk1ocen.in`
- **Python:** `python3 cyk.py < in/cyk1ocen.in`

Testy „ocen”:

`1ocen:` ciąg cyfr, nieprzesunięty;

`2ocen:` $n = 500$, $x_0 = 249$, kolejne liczby parzyste od 0;

`3ocen:` $n = 500\,000$, $x_0 = 0$, kolejne liczby nieparzyste od 1.