Zadanie: GRA
Gra w dzielniki

Olimpiada Informatyczna

XXVI OI, etap III, dzień drugi. Plik źródłowy `gra.*` Dostępna pamięć: 256 MB. *11.04.2019*

Gra w dzielniki to prosta gra polegająca na sprytnym operowaniu na liczniku x, początkowo równym 0, aby doprowadzić go do wartości równej n. W tym celu gracz będzie otrzymywał kolejne elementy pewnego nieskończonego losowego ciągu liczb o wartościach od 1 do k. Niech x oznacza bieżący stan licznika, a kolejnym otrzymanym elementem ciągu będzie y. Jeśli y jest dzielnikiem x, to można (ale nie trzeba) zwiększyć raz stan licznika x o otrzymaną wartość y.

Każdy element w ciągu jest losowany niezależnie i każda z wartości $\{1, 2, \ldots, k\}$ wypada z jednakowym prawdopodobieństwem.

Celem gry jest otrzymanie wartości licznika n, pobierając nie więcej niż M elementów ciągu losowego.

Komunikacja

To zadanie jest interaktywne. Należy napisać program, który będzie grał w *Grę w dzielniki* wiele razy, komunikując się w tym celu z dostarczoną biblioteką. Parametry n, k i M są takie same dla każdej gry w obrębie jednego testu.

W przypadku rozwiązania napisanego w języku C++ należy dołączyć nagłówek `gralib.hpp` za pomocą instrukcji

```
#include "gralib.hpp"
```

a w przypadku języka Python należy zaimportować funkcje z biblioteki `gralib` za pomocą instrukcji

```
from gralib import dajN, dajK, dajM, nastepna, zwieksz, koniec
```

Biblioteka udostępnia następujące funkcje:

- `int dajN()` – Zwraca parametr n.
- `int dajK()` – Zwraca parametr k.
- `int dajM()` – Zwraca parametr M.
- `int nastepna()` – Zwraca losową liczbę y ze zbioru $\{1, 2, \ldots, k\}$. Celem zadania jest wywołać tę funkcję co najwyżej M razy w jednej grze.
- `void zwieksz()` – Może być wywołana *co najwyżej raz* po każdym wywołaniu funkcji `nastepna`, o ile ta zwróciła element y będący dzielnikiem licznika x. Jej wywołanie powoduje zwiększenie stanu licznika x o wartość y.
- `void koniec()` – Należy wywołać tę funkcję, gdy licznik osiągnie wartość n. Spowoduje ona zainicjowanie nowej gry, czyli ustawienie licznika na zero i odświeżenie limitu wywołań M funkcji `nastepna`.

Jeśli wywołanie funkcji `zwieksz` spowoduje, że licznik przekroczy wartość n, lub zostanie wykonane przed pierwszym wywołaniem funkcji `nastepna`, lub więcej niż raz po ostatnim wywołaniu funkcji `nastepna`, lub dla liczby y niebędącej dzielnikiem bieżącego licznika x, zostanie to potraktowane jako błędna odpowiedź. Wywołanie funkcji `koniec`, gdy licznik nie jest równy n, również zostanie potraktowane jako błędna odpowiedź. Jeśli łączna liczba wywołań funkcji `nastepna` w trakcie jednej gry przekroczy limit M, zostanie to potraktowane jako błędna odpowiedź.

Ocenianie

W każdym teście do rozegrania jest dokładnie 100 gier. To znaczy, że Twój program powinien dokładnie 100 razy wywołać funkcję `koniec`. Wywołanie jakiejkolwiek funkcji po setnym wywołaniu funkcji `koniec` zostanie potraktowane jako błędna odpowiedź.

We wszystkich testach zachodzi $100 \leq n \leq 250\,000$ i $1 \leq k \leq \sqrt{n}$. Wartości parametrów n, k i M w poszczególnych testach są podane w pliku `testy.txt` w folderze `dlazaw`.

Limity czasowe obowiązujące w poszczególnych podzadaniach są opublikowane w SIO.

Testy „ocen”:

`1ocen`: $n = 100$, $k = 10$, $M = 162$;
`2ocen`: $n = 5000$, $k = 40$, $M = 5000$;
`3ocen`: $n = 50\,000$, $k = 3$, $M = 60\,000$;
`4ocen`: $n = 250\,000$, $k = 20$, $M = 144\,280$;
`5ocen`: $n = 250\,000$, $k = 500$, $M = 83\,302$.

Przykładowy przebieg programu

Uwaga: Ze względów zdroworozsądkowych, poniższy przykład (a zarazem test przykładowy) nie przestrzega ograniczenia $n \geq 100$ zadanego w sekcji „Ocenianie”.

Wywołana funkcja	**Zwrócony wynik**	**Stan licznika**	**Opis**
`dajN()`	10	0	$n = 10$, oczekiwana końcowa wartość licznika to 10
`dajK()`	5	0	$k = 5$, elementy y będą losowane ze zbioru $\{1, 2, 3, 4, 5\}$
`dajM()`	99	0	$M = 99$, maksymalna liczba wywołań funkcji `następna` to 99
`nastepna()`	4	0	$y = 4$, możliwe jest zwiększenie stanu licznika, gdyż 4 jest dzielnikiem 0
`zwieksz()`	—	4	wartość licznika została zwiększona o 4
`nastepna()`	3	4	$y = 3$, niemożliwe jest zwiększenie stanu licznika, gdyż 3 nie jest dzielnikiem 4
`nastepna()`	4	4	$y = 4$, możliwe jest zwiększenie licznika
`zwieksz()`	—	8	wartość licznika została zwiększona o 4
`nastepna()`	4	8	$y = 4$, niemożliwe jest zwiększenie stanu licznika, gdyż przekroczyłby on docelową wartość
`nastepna()`	2	8	$y = 2$, możliwe jest zwiększenie licznika
`zwieksz()`	—	10	wartość licznika osiągnęła oczekiwaną końcową wartość 10
`koniec()`	—	10	poprzednia gra została zakończona sukcesem i liczba wywołań funkcji `nastepna` nie przekroczyła M; rozpoczęcie nowej gry (wywołania funkcji `nastepna` liczone są znów od zera)
...	...	...	...
`koniec()`	—	10	setna gra została zakończona, program powinien teraz zakończyć działanie

Eksperymenty

Przykładowe **błędne** rozwiązania wraz z przykładowymi bibliotekami znajdują się w folderze `dlazaw`. Biblioteki mogą różnić się zachowaniem od tych na sprawdzaczkach i nie spełniać założeń zadania. Mają one jedynie pokazać sposób interakcji z programem.

W tym zadaniu polecenie kompilacji (w przypadku języka C++) i uruchomienia (w przypadku języka Python) jest standardowe. Trzeba jedynie zadbać o to, aby plik `gralib.hpp` (w przypadku C++) lub `gralib.py` (w przypadku Pythona) znajdował się w tym samym folderze co rozwiązanie. Dostarczony jest też przykładowy plik `makefile` generujący pliki wykonywalne `graCPP.e` i `graPY.e` z plików `gra.cpp` i `gra.py` za pomocą komendy:

```
make
```