Task: GRA
Game of divisors

XXVI OI, Stage III, Day two. Source file `gra.*` Available memory: 256 MB. *11.04.2019*

Game of divisors is a simple game in which one has to smartly operate a counter x, initially set to 0, to bring its value to n. The player is drawing successive elements of an infinite random sequence with integer values from 1 to k. Let x denote the current value of the counter, and let y be last drawn element of the sequence. If y divides x, then it is allowed (though not required) to increment the counter x by y. Regardless of the decision, y is discarded, and a new element has to be drawn if the counter value is yet to change.

Each element of the sequence is drawn independently at random with each value among $\{1, 2, \ldots, k\}$ equally likely.

The goal is to reach the counter value of n while drawing at most M elements.

Communication

This task is interactive. You are to write a program that will repeatedly play the *Game of divisors* by communicating with library provided to this end. The parameters n, k, and M are common to each game within a single test.

In case of a solution written in C++ language, the header `gralib.hpp` should be included using the instruction

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

whereas in case of Python, appropriate functions should be imported from `gralib` library using the instruction

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

The library provides the following functions:

- `int dajN()` – Returns parameter n.
- `int dajK()` – Returns parameter k.
- `int dajM()` – Returns parameter M.
- `int nastepna()` – Returns a random number y from the set $\{1, 2, \ldots, k\}$. The goal is to call this function at most M times in a single game.
- `void zwieksz()` – This function can be called *at most once* after each call to `nastepna`, and only if the latter returned an element y which divides the counter value x. Calling `zwieksz` will then increment the counter by y.
- `void koniec()` – This function should be called when the counter value reaches n. Doing so will initialize a new game, i.e., set the counter to zero and refresh the limit M of calls to `nastepna` function.

Either of the following is treated as a wrong answer:

- a call to `zwieksz` function that makes the counter value exceed n,
- a call to `zwieksz` function made before the first call to `nastepna` function,
- more than one call to `zwieksz` function after a single call to `nastepna` function,
- a call to `zwieksz` function after a call to `nastepna` function returned y that does not divide the counter value x,
- a call to `koniec` function when counter value differs from n, or
- more than M calls to `nastepna` function in a single game.

Grading

In every single test, exactly 100 games are played. This means that your program should call the `koniec` function exactly 100 times. Calling any function after the hundredth call of `koniec` is treated as a wrong answer.

In all tests, the conditions $100 \le n \le 250\,000$ and $1 \le k \le \sqrt{n}$ hold. The values of parameters n, k, and M in particular tests are specified in `testy.txt` file in the `dlazaw` directory.

Time limits for particular tests are published in SIO.

Sample grading tests:

- `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:` $n = 250\,000$, $k = 500$, $M = 83\,302$.

Sample execution

Attention: for obvious reasons, the following example (and the corresponding sample test) do not adhere to the $n \geq 100$ condition specified in "Grading" section.

Function called	**Value returned**	**Counter value**	**Description**
`dajN()`	10	0	$n = 10$, the target value of the counter is 10
`dajK()`	5	0	$k = 5$, elements y will be drawn from the set $\{1, 2, 3, 4, 5\}$
`dajM()`	99	0	$M = 99$, maximum number of calls of `nastepna` function is 99
`nastepna()`	4	0	$y = 4$, incrementing counter is allowed, since 4 divides 0
`zwieksz()`	—	4	counter incremented by 4
`nastepna()`	3	4	$y = 3$, incrementing counter is not allowed, since 3 does not divide 4
`nastepna()`	4	4	$y = 4$, incrementing counter is allowed
`zwieksz()`	—	8	counter incremented by 4
`nastepna()`	4	8	$y = 4$, incrementing counter is not allowed, since target value would be exceeded
`nastepna()`	2	8	$y = 2$, incrementing counter is allowed
`zwieksz()`	—	10	after incrementation, the counter has reached the target value of 10
`koniec()`	—	10	the game ended successfully, with at most M calls of `nastepna` function; a new game starts, with calls of `nastepna` function counted from zero
...	...	...	...
`koniec()`	—	10	after the hundredth game ended, the program should terminate

Experiments

A sample **incorrect** solution with sample libraries can be found in the `dlazaw` folder. These libraries are intended merely to demonstrate the interaction with the program, and may thus behave differently from the ones used to grade solutions.

The commands required for compiling (in case of C++) or running (in case of Python) the program are standard. But make sure that the file `gralib.hpp` (in case of C++) or `gralib.py` (in case of Python) is located in the same directory as your solution. A sample `makefile` file is also provided, which allows generating executable files `graCPP.e` and `graPY.e` out of `gra.cpp` and `gra.py` with the command:

```
make
```