Zadanie: ZER
Zera i jedynki

Olimpiada Informatyczna, dzień próbny. Plik źródłowy `zer.*` Dostępna pamięć: 256 MB.

Jest sobie pewien ciąg $a_0, \ldots, a_{n-1}$ złożony z liczb 0 i 1. Nie znasz elementów tego ciągu, możesz jednak zapytać o sumę dowolnej pary różnych elementów ciągu. Twoim zadaniem jest odgadnąć ciąg za pomocą małej liczby takich pytań.

Komunikacja

To zadanie jest interaktywne. W zadaniu komunikacja z procesem sprawdzającym odbywać się może albo przy użyciu biblioteki, albo przez standardowe wejście i wyjście. Ty wybierasz, którą opcję wolisz. **Nie należy używać obydwu metod jednocześnie.**

Przykładową implementację obydwu typów komunikacji można znaleźć w plikach dla zawodnika.

Twój program nie może otwierać żadnych plików. Program może korzystać ze standardowego wyjścia diagnostycznego (`stderr`), jednak pamiętaj, że zużywa to cenny czas. Jakiekolwiek nadmiarowe dane wypisane na standardowe wyjście mogą zostać potraktowane jako błędna odpowiedź.

Uwaga: Podane ograniczenia pamięci i czasu dotyczą tylko Twojego rozwiązania (nie wliczają procesu sprawdzającego).

Opcja 1: Komunikacja przy pomocy biblioteki

W tej opcji Twój program nie może używać standardowego wejścia i wyjścia.

C++

Na początku programu należy napisać:

- C++: `#include "zerlib.h"`

Biblioteka udostępnia następujące funkcje:

- `int daj_n()` – Funkcja daje w wyniku liczbę całkowitą n, oznaczającą długość nieznanego ciągu zer i jedynek. Funkcję tę można wywołać dowolnie wiele razy.
- `int suma(int i, int j)` – Funkcja daje w wyniku $a_i + a_j$, jeśli $0 \leq i, j < n$ oraz $i \neq j$. W przeciwnym razie jej wywołanie spowoduje błędną odpowiedź.
- `void odpowiedz(std::vector<int> a)` – Ta funkcja pozwala zgłosić wynikowy ciąg. Należy ją wykonać dokładnie raz. Przyjmuje ona tablicę $a[0..n-1]$ daną jako wektor `std::vector<int>` reprezentującą ciąg wynikowy $a[0], \ldots, a[n-1]$. Jeśli podany w funkcji ciąg będzie błędny lub jego długość będzie inna niż n, spowoduje to błędną odpowiedź. **Po wykonaniu tej funkcji należy zakończyć program.**

Python

Na początku programu należy napisać:

- Python: `from zerlib import daj_n, suma, odpowiedz`

Biblioteka udostępnia następujące funkcje:

- `daj_n() -> int` – Funkcja daje w wyniku liczbę całkowitą n, oznaczającą długość nieznanego ciągu zer i jedynek. Funkcję tę można wywołać dowolnie wiele razy.
- `suma(i : int, j : int) -> int` – Funkcja daje w wyniku $a_i + a_j$, jeśli $0 \leq i, j < n$ oraz $i \neq j$. W przeciwnym razie jej wywołanie spowoduje błędną odpowiedź.
- `odpowiedz(a : list[int])` – Ta funkcja pozwala zgłosić wynikowy ciąg. Należy ją wykonać dokładnie raz. Przyjmuje ona tablicę $a[0..n-1]$ daną jako lista, reprezentującą ciąg wynikowy $a[0], \ldots, a[n-1]$. Jeśli podany w funkcji ciąg będzie błędny lub jego długość będzie inna niż n, spowoduje to błędną odpowiedź. **Po wykonaniu tej funkcji należy zakończyć program.**

Olimpiada Informatyczna finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Opcja 2: Komunikacja przez standardowe wejście/wyjście

W pierwszym wierszu standardowego wejścia znajdować się będzie jedna liczba całkowita n, oznaczająca długość nieznanego ciągu zer i jedynek.

Aby zadać pytanie bibliotece, należy wypisać dwie liczby całkowite i i j, takie że $0 \leq i, j < n$ oraz $i \neq j$. Jeśli liczby nie będą spełniały tych warunków, spowoduje to błędną odpowiedź. Liczby należy wypisać na standardowe wyjście, oddzielone pojedynczym odstępem i zakończone znakiem nowej linii. Następnie należy wczytać jedną liczbę całkowitą ze standardowego wejścia, oznaczającą $a_i + a_j$.

Na końcu należy zgłosić wynikowy ciąg. Aby to zrobić, należy w jednym wierszu wypisać liczbę n, a w drugim ciąg n liczb rozdzielonych pojedynczymi odstępami, reprezentujący ciąg wynikowy $a[0], \ldots, a[n-1]$. Jeśli wypisany ciąg będzie błędny lub jego długość będzie inna niż n, spowoduje to błędną odpowiedź. **Po wypisaniu ciągu należy zakończyć program.**

C++, wejście/wyjście strumieniowe

Należy standardowo załączyć odpowiedni nagłówek (`#include <iostream>`). Na końcu każdego wiersza przy wypisywaniu należy używać `std::endl`. Przykładowy początek komunikacji poniżej:

```
std::cin >> n;
std::cout << i << ' ' << j << std::endl;
std::cin >> suma;
```

C++, wejście/wyjście przez `stdio`

Należy standardowo załączyć odpowiedni nagłówek (`#include <cstdio>`). Po wypisaniu każdego wiersza należy napisać `fflush(stdout)`. Przykładowy początek komunikacji poniżej:

```
scanf("%d", &n);
printf("%d %d\n", i, j);
fflush(stdout);
scanf("%d", &suma);
```

Python

Po wypisaniu każdego wiersza należy napisać `flush = True`. Przykładowy początek komunikacji poniżej:

```
n = int(input())
print(f"{i} {j}", flush = True)
suma = int(input())
```

Ocenianie

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

Podzadanie	Warunki	Liczba punktów
1	$3 \leq n \leq 1000$	50
2	$3 \leq n \leq 200\,000$	50

Biblioteka **nie musi** generować ciągu $a_0, \ldots, a_{n-1}$ na początku interakcji z Twoim programem. Może ona w trakcie interakcji zmieniać elementy ciągu, o ile nowe elementy są nadal spójne z wynikami zwróconymi przez dotychczasowe wywołania funkcji `suma`.

Jeśli m to maksymalna liczba wywołań funkcji `suma`, które Twój program wykonał w jednym przypadku testowym, to Twoje rozwiązanie otrzyma następujący procent punktów za test (odpowiednio przeskalowany w przypadku przekroczenia połowy limitu czasowego):

Liczba wywołań	Procent punktów
$m \leq n$	100% punktów za test
$m = n + 1$	80% punktów za test
$m \leq n^2 - n$	50% punktów za test
$m > n^2 - n$	0% punktów za test (werdykt *Błędna odpowiedź*)

Olimpiada Informatyczna finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2022/2023, 2023/2024, 2024/2025”.

Przykładowy przebieg programu

Poniżej przedstawiono przykładowy przebieg programu dla testu przykładowego.

Akcja	Parametry	Wynik	Opis
daj n	–	5	$n = 5$
suma	$i = 0$, $j = 1$	1	$a_0 + a_1 = 1$
suma	$i = 1$, $j = 2$	1	$a_1 + a_2 = 1$
suma	$i = 3$, $j = 4$	2	$a_3 + a_4 = 2$, a więc $a_3 = a_4 = 1$
suma	$i = 0$, $j = 3$	2	$a_0 + a_3 = 2$, a więc $a_0 = 1$, skąd wynika też, że $a_1 = 0$ i $a_2 = 1$
odpowiedź	$a = 1, 0, 1, 1, 1$	–	Prawidłowa odpowiedź z użyciem $m = 4 \leq n = 5$ pytań, 100% punktów za test

Testy przykładowe. Test 0 to test z przykładu powyżej. Poza tym:

`1ocen`: $n = 1000$, $a = \underbrace{00\ldots0}_{500}\underbrace{11\ldots1}_{500}$;

`2ocen`: $n = 200\,000$, $a = \underbrace{(01)(01)\ldots(01)}_{100\,000}$.

Dla zawodnika

W katalogu `dlazaw` znajdują się przykładowe (błędne) rozwiązania w C++ oraz Pythonie, które ilustrują obydwa modele komunikacji. Są też dostępne testy przykładowe (`zer0.in`, `zer1ocen.in`, `zer2ocen.in`), biblioteki do komunikacji oraz przykładowy program sprawdzający. Testy przykładowe są w następującym formacie:

- w pierwszym wierszu: liczba całkowita n,
- w drugim wierszu: ciąg liczb $a_0, \ldots, a_{n-1}$, rozdzielonych odstępami.

Przykładowy program sprawdzający jest inny od tego używanego w SIO. W szczególności ciąg $a_0, \ldots, a_{n-1}$ jest ustalony na początku interakcji z Twoim programem. Ponadto przykładowy program może nie sprawdzać poprawności wejścia ani argumentów wywołania funkcji. Gdy wyślesz rozwiązanie do SIO, zostanie ono sprawdzone na testach przykładowych za pomocą właściwego programu sprawdzającego.

Uwaga: W tym zadaniu nie są dostępne uruchomienia próbne w SIO ani skrypt `ocen` na komputerach.

Opcja 1: Komunikacja przy pomocy biblioteki

Pliki `zer.cpp` i `zer.py` zawierają przykładowe błędne rozwiązania demonstrujące komunikację przy pomocy biblioteki. Aby skompilować przykładowe rozwiązanie w C++, możesz użyć następującego polecenia, które tworzy plik `zer.e`:

- `g++ -O3 -static -std=c++20 zerlib.cpp zer.cpp -o zer.e`

Rozwiązanie można uruchomić np. na teście przykładowym za pomocą polecenia:

- C++: `./zer.e <zer0.in`
- Python: `python3 zer.py <zer0.in`

Opcja 2: Komunikacja przez standardowe wejście/wyjście

Pliki `zer2.cpp` i `zer2.py` zawierają przykładowe błędne rozwiązania demonstrujące komunikację przy pomocy standardowego wejścia i wyjścia. Aby skompilować przykładowe rozwiązanie w C++ oraz program sprawdzający, możesz użyć komendy `make`, która tworzy plik `zer2.e`.

Rozwiązanie można uruchomić przy pomocy skryptu `run.sh`. Komenda przyjmuje informację o skompilowanym programie oraz nazwę pliku testowego. Programy przykładowe można uruchomić na teście przykładowym `zer0.in` komendami:

- C++: `./run.sh "./zer2.e" zer0.in`
- Python: `./run.sh "python3 zer2.py" zer0.in`