Zadanie: LIC
Licytacje

Olimpiada Informatyczna

XXXIII OI, etap III, dzień pierwszy. Plik źródłowy `lic.*` Dostępna pamięć: 256 MB. *18.03.2026*

Algosia i Bajtek biorą razem udział w teleturnieju *Licytacja*. W *Licytacji* Algosia i Bajtek będą ze sobą współpracować, aby wygrać główną nagrodę. Podczas gry każde z nich dostanie po jednym ciągu liczb całkowitych długości n: Algosia dostanie ciąg liczb całkowitych $a = (a_1, a_2, \ldots, a_n)$, a Bajtek dostanie ciąg liczb całkowitych $b = (b_1, b_2, \ldots, b_n)$. Gracze znają jedynie swoje ciągi, więc Algosia nie zna ciągu b, a Bajtek nie zna ciągu a. Podczas gry oboje wiedzą, że $1 \leq a_i, b_i \leq m$ dla każdego $1 \leq i \leq n$.

W teleturnieju można wygrać W bajtalarów, gdzie:

$$W = \left\lfloor \sum_{i}^{n} \frac{\min(a_i, b_i)}{k} \right\rfloor$$

a k jest stałą znaną Algosi i Bajtkowi przed rozpoczęciem zabawy.

Jak widać, W zależy od ciągów a oraz b, które na początku gry dostają Algosia i Bajtek. Aby uniknąć zarzutów o zaniżaniu nagród w teleturnieju, organizatorzy postanowili, że ciągi a oraz b zostaną wygenerowane losowo. Szczegóły dotyczące losowości w testach znajdują się w sekcji *Ocenianie*.

Po otrzymaniu ciągów a i b, Algosia i Bajtek przystępują do licytacji. Licytacja przebiega w turach. Zaczyna Algosia, a następnie gracze wykonują ruchy na zmianę. W swoim ruchu gracz może zalicytować pewną liczbę **całkowitą** oznaczającą proponowaną wartość wygranej W. Każda kolejna zaproponowana liczba musi być **ściśle większa** od poprzednio zalicytowanej wartości. Jeśli ponadto gracz jest pewien, że zna wartość W, to może zakończyć licytację. Oznacza to, że po jego ruchu cała licytacja zostanie przerwana i oboje gracze dowiedzą się, czy W jest równe tej zalicytowanej wartości. Jeśli tak, to Algosia i Bajtek wygrywają nagrodę, a jeśli nie, to przegrywają. Jeśli gracz nie zakończy licytacji w danym ruchu, to po tym ruchu jego partner dowiaduje się, jaka jest nowa zalicytowana wartość.

Twoim zadaniem jest zaimplementowanie strategii, która będzie symulować grę Algosi i Bajtka. Ze względu na losowość ciągów a i b Twój program w każdym przypadku testowym program wykonuje jedną *rozgrywkę*, która składa się z 40 gier w *Licytację*. Wynik tejże rozgrywki będzie zależał od liczby wygranych gier. Szczegóły dotyczące punktacji znajdują się w sekcji *Ocenianie*.

Komunikacja

To zadanie jest interaktywne. Należy napisać program, który będzie grał jako Algosia i Bajtek. Podczas sprawdzania zostanie on uruchomiony w dwóch **osobnych** procesach jednocześnie. Jeden z tych procesów będzie reprezentował Algosię, a drugi Bajtka. Komunikacja z programem sprawdzającym odbywa się przy pomocy udostępnionej biblioteki. Twój program nie może używać standardowego wejścia i wyjścia.

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

C++

Na początku programu należy napisać:

- `#include "liclib.h"`

Biblioteka udostępnia następujące funkcje:

- `int IleGier()` – Zwraca liczbę gier t w *Licytację*, które będą rozgrywane. W każdym przypadku testowym ta wartość jest równa $t = 40$ i nie zmienia się w trakcie rozgrywki.
- `int KimJestem()` – Zwraca 0, jeśli wywołujący program reprezentuje Algosię, lub 1, jeśli reprezentuje Bajtka.
- `std::vector<int> DajCiag()` – Zwraca ciąg liczb całkowitych, który został przydzielony wywołującemu programowi na początku gry. Jeśli program reprezentuje Algosię, to zwrócony ciąg to a, a jeśli reprezentuje Bajtka, to zwrócony ciąg to b.
- `int DajM()` – Zwraca parametr m z treści zadania.
- `int DajK()` – Zwraca parametr k z treści zadania.

Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

- `std::pair<int, bool> RuchPartnera()` – Zwraca parę `(x, koniec)`, gdzie:
  - x jest wartością zalicytowaną przez partnera w jego ostatnim ruchu,
  - `koniec` informuje, czy partner zakończył licytację w tej grze.

  Jeśli `koniec` jest równy `true`, licytacja w tej grze jest zakończona i nie należy wykonywać dalszych ruchów.
- `void Licytuj(int stawka, bool koniec)` – Funkcja służy do zalicytowania liczby całkowitej `stawka`. Liczba ta musi być ściśle większa niż wartość poprzednio zalicytowana przez partnera (lub większa niż 0 w przypadku pierwszego ruchu Algosi).
  Jeśli parametr `koniec` jest równy `true`, gracz ogłasza zakończenie licytacji i twierdzi, że rzeczywista wartość W jest równa `stawka`. Po wykonaniu takiego ruchu żaden z graczy nie powinien już wykonywać kolejnych ruchów w tej grze (ponadto partner powinien odebrać tą parę wartości przed zakończeniem gry).

Wszystkie funkcje, za wyjątkiem funkcji `RuchPartnera` oraz `Licytuj`, można wywoływać w dowolnym momencie, dowolnie wiele razy.

Python

Na początku programu należy napisać:

- `from liclib import IleGier, KimJestem, DajCiag, DajM, DajK, RuchPartnera, Licytuj`

Biblioteka udostępnia następujące funkcje:

- `IleGier() -> int` – Zwraca liczbę gier t w *Licytację*, które będą rozgrywane. W każdym przypadku testowym ta wartość jest równa $t = 40$ i nie zmienia się w trakcie rozgrywki.
- `KimJestem() -> int` – Zwraca 0, jeśli wywołujący program reprezentuje Algosię, lub 1, jeśli reprezentuje Bajtka.
- `DajCiag() -> list[int]` – Zwraca ciąg liczb całkowitych, który został przydzielony wywołującemu programowi na początku gry. Jeśli program reprezentuje Algosię, to zwrócony ciąg to a, a jeśli reprezentuje Bajtka, to zwrócony ciąg to b.
- `DajM() -> int` – Zwraca parametr m z treści zadania.
- `DajK() -> int` – Zwraca parametr k z treści zadania.
- `RuchPartnera() -> tuple[int, bool]` – Zwraca parę `(x, koniec)`, gdzie:
  - x jest wartością zalicytowaną przez partnera w jego ostatnim ruchu,
  - `koniec` informuje, czy partner zakończył licytację w tej grze.

  Jeśli `koniec` jest równy `true`, licytacja w tej grze jest zakończona i nie należy wykonywać dalszych ruchów.
- `Licytuj(stawka: int, koniec: bool) -> None` – Funkcja służy do zalicytowania liczby całkowitej `stawka`. Liczba ta musi być ściśle większa niż wartość poprzednio zalicytowana przez partnera (lub większa niż 0 w przypadku pierwszego ruchu Algosi).
  Jeśli parametr `koniec` jest równy `true`, gracz ogłasza zakończenie licytacji i twierdzi, że rzeczywista wartość W jest równa `stawka`. Po wykonaniu takiego ruchu żaden z graczy nie powinien już wykonywać kolejnych ruchów w tej grze (ponadto partner powinien odebrać tą parę wartości przed zakończeniem gry).

Wszystkie funkcje, za wyjątkiem funkcji `RuchPartnera` oraz `Licytuj`, można wywoływać w dowolnym momencie, dowolnie wiele razy.

Dodatkowe informacje

W każdej rozgrywce wartości $t = 40, n, m$ oraz k są stałe. Wartości n, m oraz k mogą się zmieniać pomiędzy testami. Ponadto w każdym wywołaniu programu wartość zwracana przez funkcję `KimJestem()` jest stała.

Gra zostaje uznana za zakończoną dla danego gracza wtedy i tylko wtedy, gdy:

- gracz wykona funkcję `Licytuj` z argumentem `koniec` ustawionym na `true`, lub
- w wyniku wywołania funkcji `RuchPartnera` gracz dowie się, że partner zakończył licytację.

Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

Po zakończeniu gry funkcja `DajCiag` zacznie zwracać ciąg odpowiadający kolejnej grze dla danego gracza.

W tym zadaniu kolejność wywołań funkcji `RuchPartnera` oraz `Licytuj` jest istotna. Funkcje te muszą być wywoływane **naprzemiennie** – żadna z nich nie może zostać wywołana dwa razy z rzędu.

Jeśli program reprezentuje Algosię, to pierwszym ruchem gracza powinno być wywołanie funkcji `Licytuj`. Jeśli program reprezentuje Bajtka, to pierwszym ruchem gracza powinno być wywołanie funkcji `RuchPartnera`.

Ponadto każdy z graczy z osobna może wykonać funkcję `Licytuj` co najwyżej 1 000 000 razy. Po wykonaniu funkcji `Licytuj` po raz 1 000 000 program reprezentujący gracza powinien od razu się zakończyć (nawet jeśli nie rozegrał do końca wszystkich gier).

Niezastosowanie się do powyższych reguł, wywołanie funkcji z niepoprawnymi argumentami lub korzystanie ze standardowego wejścia i wyjścia może skutkować otrzymaniem werdyktu „błędna odpowiedź” albo „przekroczenie limitu czasu”.

Przykładowy przebieg jednej rozgrywki

Opisana sytuacja jest poglądowa i ma na celu pokazanie oczekiwanej kolejności wykonywania akcji przez program zawodnika. Twój program nie musi wykonywać się poprawnie na przedstawionym poniżej przykładzie. Załóżmy, że $t = 1$, $n = 3$, $m = 20$, $k = 4$, $a = [10, 3, 15]$ i $b = [14, 17, 5]$. W tym przypadku wartość wygranej, którą gracze chcą wylicytować, wynosi:

$$W = \left\lfloor \frac{\min(10, 14) + \min(3, 17) + \min(15, 5)}{4} \right\rfloor = \left\lfloor \frac{18}{4} \right\rfloor = 4.$$

Przebieg działania programu z komunikacją przy wykorzystaniu biblioteczki dla obojga graczy dla takich danych może być następujący:

Gracz	Akcja	Argumenty	Zwrócona wartość	Komentarz
Algosia	IleGier	–	1	Algosia dowiedziała się, że odbędzie się jedna gra.
Bajtek	IleGier	–	1	Bajtek dowiedział się, że odbędzie się jedna gra
Algosia	KimJestem	–	0	Program będzie grał jako Algosia.
Bajtek	KimJestem	–	1	Program będzie grał jako Bajtek.
Algosia	DajCiag	–	$[10, 3, 15]$	Ciąg Algosi to $[10, 3, 15]$.
Bajtek	DajCiag	–	$[14, 17, 5]$	Ciąg Bajtka to $[14, 17, 5]$.
Algosia	Licytuj	2, `false`	–	Algosia licytuje 2 i nie kończy rozgrywki.
Bajtek	RuchPartnera	–	2, `false`	Bajtek dowiedział się, że Algosia zalicytowała 2 i nie zakończyła rozgrywki.
Bajtek	Licytuj	3, `false`	–	Bajtek licytuje 3 i nie kończy rozgrywki.
Algosia	RuchPartnera	–	3, `false`	Algosia dowiedziała się, że Bajtek zalicytował 3 i nie zakończył rozgrywki.
Algosia	Licytuj	4, `true`	–	Algosia magicznym cudem zgaduje, że odpowiedź powinna wynosić 4, licytuje tą wartość i kończy rozgrywkę. Okazuje się to być trafny strzał. Algosia kończy komunikację, gdyż tylko jedna gra jest rozgrywana.
Bajtek	RuchPartnera	–	4, `true`	Bajtek dowiedział się, że Algosia zalicytowała 4 i zakończyła rozgrywkę. Bajtek kończy komunikację.

Testy przykładowe. W każdym z testów przykładowych będzie zachodzić $t = 2$. Ciągi Algosi i Bajtka będą wygenerowane w sposób pseudolosowy, jak w testach właściwych. Testy przykładowe mają następujące parametry:

`0a:` $n = 2, k = 1, m = 10\,000$.

`0b:` $n = 1000, m = 100\,000, k = 1\,500$.

`0c:` $n = 1000, m = 100\,000, k = 5\,555$.

`0d:` $n = 1000, m = 100\,000, k = 7\,000$.

Sfinansowano ze środków Ministra Edukacji

Ministerstwo Edukacji Narodowej

Ocenianie

Zestaw testów dzieli się na następujące podzadania.

Podzadanie	Ograniczenia	Punkty
1	$n = 2$, $m = 10\,000$, $k = 1$	10
2	$n = 10$, $m = 25\,000$, $k = 200$	10
3	$n = 10$, $m = 25\,000$, $k = 300$	15
4	$n = 1\,000$, $m = 100\,000$, $k = 1\,500$	10
5	$n = 1\,000$, $m = 100\,000$, $k = 3\,500$	10
6	$n = 1\,000$, $m = 100\,000$, $k = 5\,555$	15
7	$n = 1\,000$, $m = 100\,000$, $k = 6\,000$	10
8	$n = 1\,000$, $m = 100\,000$, $k = 6\,500$	10
9	$n = 1\,000$, $m = 100\,000$, $k = 7\,000$	10

W każdym podzadaniu jest dokładnie jeden test. *Rozgrywka* w tym teście składa się z $t = 40$ gier. Ciągi a i b w każdym teście zostały wygenerowane w sposób pseudolosowy. Dokładniej, w każdej grze każdy element ciągu a i ciągu b jest generowany niezależnie z rozkładu jednostajnego. To oznacza, że szansa na $a_i = x$ jest równa dokładnie $\frac{1}{m}$ dla każdego i oraz całkowitego x spełniającego $1 \leq x \leq m$ (analogiczne warunki spełnia ciąg b).

Punktacja pojedynczego testu zależy od liczby wygranych gier. Załóżmy, że Algosia i Bajtek wygrali p gier. Wówczas wynik za ten test wynosi:

$$\min\left(100, \left\lfloor \frac{p^2}{4} \right\rfloor\right)$$

Na przykład, dla $p = 10$ wynik to 25, a dla $p = 20$ wynik to 100.

Dla zawodnika

W katalogu `dlazaw` znajdują się przykładowe (błędne) rozwiązania w językach C++ oraz Python. Są też dostępne biblioteki do komunikacji oraz przykładowy program sprawdzający.

- `lic.cpp` – przykładowe rozwiązanie demonstrujące komunikację przy pomocy `liclib.h`.
- `lic.py` – przykładowe rozwiązanie demonstrujące komunikację przy pomocy `liclib.py`.

Aby skompilować przykładowe rozwiązania w C++ oraz program sprawdzający, możesz użyć komendy `make`, która tworzy pliki `por.e`.

Program można uruchomić przy pomocy skryptu `run.sh`. Dla przykładu wszystkie powyższe programy można uruchomić komendami:

- `./run.sh "./lic.e"`
- `./run.sh "python3 lic.py"`

Uruchomiony program wczytuje dane z pliku `input.in` w następującym formacie:

- w pierwszym wierszu: pięć liczb całkowitych t, g, n, m, k, o znaczeniu opisanym w sekcji dotyczącej komunikacji.
- w kolejnych $2t$ wierszach powinny znajdować się opisy kolejnych gier. Opis każdej gry składa się dwóch wierszy, z których każdy zawiera ciąg n liczb całkowitych z zakresu $[1, m]$. Pierwszy wiersz opisuje ciąg Algosi, a drugi ciąg Bajtka w danej grze.

Liczby powinny być rozdzielone pojedynczymi odstępami.

Przykładowy program sprawdzający jest inny od tego używanego w SIO. W szczególności 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.