

W zoo mieszka $2N$ kameleonów ponumerowanych od 1 do $2N$. Dokładnie N z nich jest płci X, a pozostałe są płci Y. Każdy kameleon ma swój pierwotny kolor. Kameleony płci X mają parami różne pierwotne kolory. Dla każdego z nich istnieje dokładnie jeden kameleon płci Y o tym samym pierwotnym kolorze.

Każdy kameleon kocha dokładnie jednego kameleona przeciwnej płci, o innym pierwotnym kolorze. Żadne dwa kameleony nie kochają tego samego kameleona. Ani płcie, ani kolory, ani relacje miłości nie są znane Twojemu programowi.

Możesz organizować spotkania dowolnych zbiorów kameleonów. Jeśli na spotkaniu są kameleon s i kameleon t , którego s kocha, to s przyjmuje *pierwotny* kolor kameleona t . Jeśli t nie jest obecny, kameleon s zachowuje swój pierwotny kolor. Zmiany zachodzą jednocześnie i nie zależą od poprzednich spotkań. Dla każdego spotkania poznajesz jedynie liczbę różnych kolorów obecnych na nim kameleonów.

Znajdź wszystkie N par kameleonów o jednakowym pierwotnym kolorze, organizując najwyżej 20 000 spotkań. Obowiązuje $2 \leq N \leq 500$.

Interfejs programu

To zadanie biblioteczne. Zgłoś jeden plik C++ zawierający `#include "chameleon.h"` oraz implementację funkcji:

```
void Solve(int N);
```

Biblioteka wywoła tę funkcję dokładnie raz w pojedynczym uruchomieniu programu. Nie implementuj funkcji `main`. Twój program może korzystać z funkcji:

```
int Query(const std::vector<int>& p);  
void Answer(int a, int b);
```

`Query(p)` organizuje spotkanie kameleonów z listy p i zwraca liczbę różnych kolorów. Numery na liście muszą być różne i należeć do $[1, 2N]$. Lista może być pusta; wtedy wynikiem jest 0. Łącznie możesz wywołać tę funkcję najwyżej 20 000 razy.

`Answer(a,b)` zgłasza parę kameleonów o jednakowym pierwotnym kolorze. Numery a i b muszą być różne i należeć do $[1, 2N]$. Każdy kameleon musi wystąpić w dokładnie jednym zgłoszeniu. Przed zakończeniem `Solve` należy wywołać `Answer` dokładnie N razy; kolejność par jest dowolna.

Naruszenie któregokolwiek warunku oznacza błędną odpowiedź. Program nie może czytać standardowego wejścia, pisać na standardowe wyjście ani korzystać z innych plików. Informacje diagnostyczne można wypisywać na standardowe wyjście błędów (`std::cerr`).

Przykładowa komunikacja

Dla $N = 4$ możliwa jest następująca komunikacja:

Wywołanie	Wynik
<code>Query({})</code>	0
<code>Query({6,2})</code>	2
<code>Query({8,1,6})</code>	2
<code>Query({7,1,3,5,6,8})</code>	4
<code>Query({8,6,4,1,5})</code>	3
<code>Answer(6,4)</code>	—
<code>Answer(7,8)</code>	—
<code>Answer(2,1)</code>	—
<code>Answer(3,5)</code>	—

Podane pary są poprawne dla przykładowego układu kameleonów.

Testowanie lokalne

Pliki `grader.cpp` i `chameleon.h` są dostępne w materiałach do zadania. Umieść je obok swojego pliku `chameleon.cpp` i skompiluj:

```
g++ -std=c++17 -O2 grader.cpp chameleon.cpp -o chameleon
./chameleon < sample-01.in
```

Wejście lokalnego gradera ma cztery wiersze: N , następnie $2N$ liczb Y_i (płcie: 0 oznacza X, 1 oznacza Y), $2N$ liczb C_i (pierwotne kolory od 1 do N) i $2N$ liczb L_i (numery kochanych kameleonów). To format wyłącznie dla biblioteki, a nie wejście Twojego rozwiązania. Grader po sukcesie wypisuje `CHA_OK`, liczbę zapytań, N i znalezione pary. W razie błędu wypisuje `CHA_WRONG` i kod błędu. Na Szkopule biblioteka jest dołączana automatycznie: zgłaszasz tylko swój plik.

Ocenianie

Limit czasu wynosi 3 sekundy, a limit zapytań 20 000 w każdym podzadaniu.

Podzadanie	Ograniczenia	Limit czasu	Punkty
1	Miłość jest wzajemna: $L_{L_i} = i$.	3 s	4
2	$N \leq 7$.	3 s	20
3	$N \leq 50$.	3 s	20
4	Kameleony $1, \dots, N$ są płci X.	3 s	20
5	Brak dodatkowych ograniczeń.	3 s	36

Podzadania są niezależne. Punkty za podzadanie otrzymasz po zaliczeniu wszystkich jego testów. Ukryte dane nie zmieniają się w trakcie uruchomienia.

Źródło: JOI 2019/2020, Spring Training Camp, dzień 2, *Chameleon's Love*. Autor: Sora Todaka. Polska adaptacja do OKI Advanced. Treść na licencji CC BY-SA 4.0; oryginał: www.ioi-jp.org.