Miłość kameleonów

Japoński Obóz Wiosenny 2020, dzień 2
21 marca 2020

Kod zadania: **chameleon**
Limit czasu: **2 s**
Limit pamięci: **537 MB**

W pewnym zoo w mieście Joi mieszka $2N$ kameleonów, ponumerowanych dla wygody od 1 do $2N$. Spośród nich, dokładnie N ma płeć X, a pozostałych N – płeć Y. Każdy z kameleonów ma również swój **oryginalny kolor** – o oryginalnych kolorach wiadomo, że:

- wszystkie oryginalne kolory kameleonów płci X są różne;
- dla każdego kameleona płci X istnieje dokładnie jeden kameleon płci Y mający ten sam oryginalny kolor.

Właśnie przyszła wiosna, a dla kameleonów to czas wielkich, romantycznych przygód. Każdy kameleon zakochał się w pewnym innym kameleonie – miłość nie musi być jednak odwzajemniona, wiadomo tylko, że:

- każdy kameleon kocha kameleona innej płci niż on sam;
- kameleon zawsze kocha kameleona o innym oryginalnym kolorze;
- żadne dwa kameleony nie wybrały tego samego obiektu uczuć.

Pod wpływem uczuć, kameleony zaczęły zmieniać kolory, przez co pracownicy zoo pogubili się trochę w sytuacji. Twoim zadaniem jest dyskretnie się w niej zorientować. Możesz w tym celu zwoływać spotkania wybranych przez siebie grup kameleonów. Dla każdego kameleona s wiesz, że przybierze na takim spotkaniu kolor według następujących reguł:

- jeśli obiektem uczuć s jest t, oraz t również jest obecny na spotkaniu, wtedy s przybierze oryginalny kolor kameleona t.
- jeśli obiektem uczuć s jest t, ale t nie ma na spotkaniu, wtedy s będzie miał swój własny oryginalny kolor.

Może się oczywiście zdarzyć, że według tych reguł ten sam kameleon może mieć różne kolory na różnych spotkaniach. Ponieważ niezbyt dobrze rozróżniasz kameleony, i nie znasz zbyt wielu nazw kolorów, jedyne co wynosisz ze spotkania, to wiedza, ile było na nim reprezentowanych różnych kolorów.

Wyznacz, organizując co najwyżej $20\,000$ spotkań, które pary kameleonów różnej płci miały oryginalnie ten sam kolor.

Implementacja

Do oceny przesyłasz jeden plik, który musi nazywać się `chameleon.cpp` i załączać na początku nagłówek `chameleon.h`. Twój program nie powinien czytać ze standardowego wejścia ani pisać na standardowe wyjście. Musi implementować funkcję:

- `void Solve(int N)`

  Dla każdego testu sprawdzarka uruchomi tę funkcję dokładnie raz.

  - Parametr `N` odpowiada liczbie N z treści zadania, czyli liczbie kameleonów mających płeć X.

Twój program ma dostęp do następujących dwóch funkcji:

★ `int Query(const std::vector<int> &p)`

Wywołanie tej funkcji odpowiada zwołaniu spotkania kameleonów.

- ◇ Jako parametr `p` podajesz listę kameleonów, które zaprosisz na spotkanie.
- ◇ Funkcja zwraca liczbę całkowitą – liczbę różnych kolorów, jakie zobaczysz na spotkaniu.
- ◇ Każdy element wektora `p` powinien być liczbą całkowitą między 1 a $2N$ włącznie. Jeśli ten warunek nie będzie spełniony, Twój program dostanie komunikat **Wrong Answer [1]**.

- ◇ Elementy wektora `p` powinny być różne. Jeśli ten warunek nie będzie spełniony, Twój program dostanie komunikat **Wrong Answer [2]**.
- ◇ Funkcję `Query` możesz wykonać co najwyżej $20\,000$ razy. Jeśli ten warunek nie będzie spełniony, Twój program dostanie komunikat **Wrong Answer [3]**.

★ `void Answer(int a, int b)`

Wywołując tę funkcję, odpowiadasz że dwa kameleony mają ten sam kolor.

- ◇ Parametry `a` i `b` to numery kameleonów, o których uważasz, że miały oryginalnie ten sam kolor.
- ◇ Musi zachodzić $1 \le a \le 2N$ i $1 \le b \le 2N$. Jeśli ten warunek nie będzie spełniony, Twój program dostanie komunikat **Wrong Answer [4]**.
- ◇ Żadna wartość a ani b nie powinna pojawić się w wywołaniach funkcji więcej niż raz. Jeśli ten warunek nie będzie spełniony, Twój program dostanie komunikat **Wrong Answer [5]**.
- ◇ Jeśli Twój program nie zgadł prawidłowo, i kameleony a oraz b miały różne kolory, Twój program dostanie komunikat **Wrong Answer [6]**.
- ◇ Twój prorgam powinien wywołać funkcję `Answer` dokładnie N razy. Jeśli w momencie zakończenia funkcji `Solve` liczba wywołań `Answer` jest inna niż N, Twój program dostanie komunikat **Wrong Answer [7]**.

Uwagi

- Twój program może na własny użytek implementować też inne funkcje, może też używać globalnych zmiennych.
- Twój program nie może używać standardowego wejścia i wyjścia, ani komunikować się z żadnymi innymi plikami poza wymienionymi. Wolno mu jednak wypisywać pomocnicze informacje na wyjście błędów (`stderr`).

Kompilacja, uruchomienie przykładowe

Ze strony konkursu możesz pobrać spakowane archiwum, które zawiera przykładową, pomocniczą sprawdzarkę dla Twojego rozwiązania. Archiwum zawiera też przykładowy plik źródłowy programu.

Sprawdzarka to plik `grader.cpp`. Aby przetestować swój program, umieść w jednym katalogu pliki `grader.cpp`, `chameleon.cpp`, oraz `chameleon.h` i uruchom poniższą komendę, żeby skompilować swój program:

```
g++ -std=gnu++14 -O2 -o grader grader.cpp chameleon.cpp
```

Jeśli kompilacja się uda, stworzy się wykonywalny plik `grader`.

Pamiętaj, że właściwa sprawdzarka jest inna niż dostarczona przykładowa. Ta przykładowa uruchamia się jako pojedynczy proces, który czyta dane ze standardowego wejścia i wypisuje na standardowe wyjście.

Wejście przykładowej sprawdzarki

Przykładowa sprawdzarka czyta ze standardowego wejścia następujące dane:

N
$Y_1 \cdots Y_{2N}$
$C_1 \cdots C_{2N}$
$L_1 \cdots L_{2N}$

Liczba Y_i $(1 \le i \le 2N)$ oznacza płeć kameleona i, równą 0 lub 1 (0 jeśli płeć to X, 1 jeśli Y).
Liczba C_i $(1 \le i \le 2N)$ to oryginalny kolor kameleona i, równy co najmniej 1 i co najwyżej N.
Liczba L_i $(1 \le i \le 2N)$ to numer kameleona, którego kocha kameleon i.

Wyjście przykładowej sprawdzarki

Jeśli wykonanie programu zakończy się bez błędu, przykładowa sprawdzarka wypisze na standardowe wyjście co następuje:

- Jeśli odpowiedź Twojego programu jest poprawna, sprawdzarka wypisze liczbę wywołań funkcji `Query` jako "`Accepted: liczba`" (np. "`Accepted: 100`").
- Jeśli odpowiedź będzie nieprawidłowa, sprawdzarka wypisze rodzaj błędu jako "`Wrong Answer [numer błędu]`", np. "`Wrong Answer [1]`".

Nawet jeśli Twój program zrobił więcej niż jeden z możliwych błędów, na wyjście wypisze się tylko jeden.

Ograniczenia

Wszystkie dane wejściowe spełniają podane poniżej ograniczenia. (Znaczenie Y, C, and L jest takie samo, jak w sekcji **Wejście przykładowej sprawdzarki**).

- $2 \leq N \leq 500$.
- $0 \leq Y_i \leq 1$ $(1 \leq i \leq 2N)$.
- $1 \leq C_i \leq N$ $(1 \leq i \leq 2N)$.
- Dla każdego j $(1 \leq j \leq N)$ istnieje dokładnie jedno i $(1 \leq i \leq 2N)$ spełniające $Y_i = 0$ oraz $C_i = j$.
- Dla każdego j $(1 \leq j \leq N)$ istnieje dokładnie jedno i $(1 \leq i \leq 2N)$ spełniające $Y_i = 1$ oraz $C_i = j$.
- $1 \leq L_i \leq 2N$ $(1 \leq i \leq 2N)$.
- $Y_i \neq Y_{L_i}$ $(1 \leq i \leq 2N)$.
- $C_i \neq C_{L_i}$ $(1 \leq i \leq 2N)$.
- $L_k \neq L_l$ $(1 \leq k < l \leq 2N)$.

Podzadania

1. (4 punkty) $L_{L_i} = i$ $(1 \leq i \leq 2N)$.
2. (20 punktów) $N \leq 7$.
3. (20 punktów) $N \leq 50$.
4. (20 punktów) $Y_i = 0$ $(1 \leq i \leq N)$.
5. (36 punktów) Brak dodatkowych warunków.

Przykładowa komunikacja

Poniżej podane jest przykładowe wejście dla przykładowej sprawdzarki, i odpowiadające mu możliwe wywołania funkcji.

Przykładowe wejście 1	Wywołania		
	Wywołanie	Wywołanie	Zwracana wartość
4 1 0 1 0 0 1 1 0 4 4 1 2 1 2 3 3 4 3 8 7 6 5 2 1	`Solve(4)`		
		`Query([])`	`0`
		`Query([6, 2])`	`2`
		`Query([8, 1, 6])`	`2`
		`Query([7, 1, 3, 5, 6, 8])`	`4`
		`Query([8, 6, 4, 1, 5])`	`3`
		`Answer(6, 4)`	
		`Answer(7, 8)`	
		`Answer(2, 1)`	
		`Answer(3, 5)`	

Wśród plików, które możesz pobrać ze strony zawodów, `sample-02.txt` spełnia ograniczenia Podzadania 1, a `sample-03.txt` spełnia ograniczenia Podzadania 4.