Zadanie: RED
Redukcja połączeń

XXXII OI, etap III, dzień drugi. Plik źródłowy `red.*` Dostępna pamięć: 256 MB. *27.03.2025*

Bajtocy został właśnie mianowany Naczelnym Architektem Bajtocji. Bajtocja składa się z n ($1 \leq n \leq 200\,000$) miast połączonych m ($1 \leq m \leq 300\,000$) dwukierunkowymi drogami, z których każda ma swój koszt utrzymania będący dodatnią liczbą całkowitą. Miasta oraz drogi są ponumerowane odpowiednio od 0 do $n-1$ oraz od 0 do $m-1$. Wiadomo, że sieć dróg umożliwia przejście między dwoma dowolnymi miastami, a koszty utrzymania dróg są parami różne. Każda para miast jest połączona co najwyżej jedną drogą. Żadna droga nie łączy miasta z samym sobą.

Wraz z jego dość nieoczekiwanym awansem Bajtocemu przyszło niestety trudne zadanie: król Bajtazar chciałby zamknąć niektóre drogi tak, aby wciąż było możliwe przejechanie między dowolnymi dwoma miastami. Oczywiście zależy mu na zminimalizowaniu sumy kosztów utrzymania niezamkniętych dróg. Bajtocy został poproszony o wyznaczenie, które drogi należy pozostawić.

Bajtocy niestety nie zna kosztów utrzymania poszczególnych dróg. Może jednak zadawać pytania swoim doradcom. Każde z pytań jest jednego z dwóch typów. Pytanie typu *niezależne* to porównanie kosztów dwóch dróg, które nie mają wspólnych końców. Pytanie typu *gwiazda* to wyznaczenie, wśród podanych dróg o wspólnym końcu, która z nich ma najmniejszy koszt. Pomóż Bajtocemu wyznaczyć, które drogi należy pozostawić, korzystając wyłącznie z zapytań obu typów. Liczba punktów, które uzyska Twoje rozwiązanie, będzie zależeć od liczby zadanych pytań każdego typu – patrz sekcja Ocenianie.

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. Celowe próby wpłynięcia na wewnętrzne działanie biblioteki oceniającej są zakazane.

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:

```
#include "redlib.h"
```

Biblioteka udostępnia następujące funkcje:

- `int DajN()` – Wynikiem funkcji jest liczba n oznaczająca liczbę miast w Bajtocji.
- `std::vector<std::pair<int,int> > DajDrogi()` – Zwraca m par liczb opisujących drogi w Bajtocji. Każda para składa się z dwóch liczb całkowitych x i y ($0 \leq x, y < n$, $x \neq y$) oznaczających miasta połączone drogą. Drogi są ponumerowane w kolejności zwróconej przez tę funkcję. Pamiętaj, że **koszty utrzymania dróg są parami różne**.
- `int Niezalezne(int a, int b)` – Porównuje drogi o numerach a oraz b ($0 \leq a, b < n$) niemające wspólnego końca; zwraca -1, jeśli droga a jest tańsza, oraz 1, jeśli droga b jest tańsza.
- `int Gwiazda(std::vector<int> t)` – Porównuje zbiór dróg o wspólnym końcu; wektor t powinien zawierać numery dróg, a wynikiem jest numer najtańszej drogi w tym zbiorze.
- `void Wynik(std::vector<int> t)` – Przyjmuje zbiór numerów dróg, które spełniają warunek z treści zadania. **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”.

Python

Na początku programu należy napisać:

```
from redlib import DajN, DajDrogi, Niezalezne, Gwiazda, Wynik
```

Biblioteka udostępnia następujące funkcje:

- `DajN() -> int` – Wynikiem funkcji jest liczba n oznaczająca liczbę miast w Bajtocji.
- `DajDrogi() -> list[tuple[int, int]]` – Zwraca m par liczb opisujących drogi w Bajtocji. Każda para składa się z dwóch liczb całkowitych x i y ($0 \leq x, y < n$, $x \neq y$) oznaczających miasta połączone drogą. Drogi są ponumerowane w kolejności zwróconej przez tę funkcję. Pamiętaj, że **koszty utrzymania dróg są parami różne**.
- `Niezalezne(a : int, b : int) -> int` – Porównuje drogi o numerach a oraz b ($0 \leq a, b < n$) niemające wspólnego końca; zwraca -1, jeśli droga a jest tańsza, oraz 1, jeśli droga b jest tańsza.
- `Gwiazda(t : list[int]) -> int` – Porównuje zbiór dróg o wspólnym końcu; lista t powinna zawierać numery dróg, a wynikiem jest numer najtańszej drogi w tym zbiorze.
- `Wynik(t : list[int])` – Przyjmuje zbiór numerów dróg, które spełniają warunek z treści zadania. **Po wykonaniu tej funkcji należy zakończyć program.**

Ogólne uwagi

Użycie którejkolwiek funkcji z niepoprawnymi argumentami będzie skutkować otrzymaniem werdyktu „błędna odpowiedź". Za niepoprawne przyjmujemy:

- podanie numeru drogi spoza zbioru $\{0, 1, \ldots, m-1\}$.
- porównanie dróg o wspólnym końcu (w tym drogi z samą sobą) za pomocą funkcji `Niezalezne`,
- wywołanie funkcji `Gwiazda` ze zbiorem dróg, które nie mają wspólnego końca,
- wywołanie funkcji `Gwiazda` z pustym zbiorem dróg,
- wywołanie funkcji `Wynik` ze zbiorem dróg, w którym jakaś droga występuje więcej niż raz.

Opcja 2: Komunikacja przez standardowe wejście/wyjście

W pierwszym wierszu standardowego wejścia znajdować się będą dwie liczby całkowite n i m, oznaczające liczbę miast oraz liczbę dróg w Bajtocji. Dalej na wejściu znajdzie się m wierszy opisujących drogi w Bajtocji. Każdy z tych wierszy zawiera dwie liczby całkowite x i y ($0 \leq x, y < n$, $x \neq y$) oznaczające miasta połączone drogą. Drogi są ponumerowane w kolejności zwróconej przez tę funkcję. Pamiętaj, że **koszty utrzymania dróg są parami różne**.

Aby zadać pytanie typu *niezależne*, należy wypisać wiersz zawierający kolejno liczbę 1 oraz dwie liczby całkowite a i b ($0 \leq a, b < n$). Takie pytanie porównuje drogi o numerach a oraz b niemające wspólnego końca. Następnie należy wczytać jedną liczbę całkowitą ze standardowego wejścia; będzie ona równa -1, jeśli droga a jest tańsza, oraz 1, jeśli droga b jest tańsza.

Aby zadać pytanie typu *gwiazda*, należy wypisać wiersz zawierający kolejno liczbę 2, liczbę dróg k oraz k numerów dróg. Takie pytanie porównuje k dróg o podanych numerach. Następnie należy wczytać jedną liczbę całkowitą ze standardowego wejścia; będzie to numer najtańszej spośród podanych dróg.

Na końcu należy wypisać wiersz zawierający kolejno liczbę 3, liczbę dróg k oraz k numerów dróg. Ma to być zbiór numerów dróg, które spełniają warunek z treści zadania.

Zadanie któregokolwiek pytania z niepoprawnymi argumentami będzie skutkować otrzymaniem werdyktu „błędna odpowiedź". Za niepoprawne przyjmujemy:

- podanie numeru drogi spoza zbioru $\{0, 1, \ldots, m-1\}$.
- porównanie dróg o wspólnym końcu (w tym drogi z samą sobą) za pomocą pytania typu *niezależne*,
- zadanie pytania typu *gwiazda* ze zbiorem dróg, które nie mają wspólnego końca,
- zadanie pytania typu *gwiazda* z pustym zbiorem dróg,
- podanie odpowiedzi ze zbiorem dróg, w którym jakaś droga występuje więcej niż raz.

C++, wejście/wyjście strumieniowe

Należy standardowo załączyć odpowiedni nagłówek (`#include <iostream>`). Na końcu każdego wiersza przy wypisywaniu należy używać `std::endl`. Przykładowe pytanie typu *niezależne* wygląda następująco:

```
std::cout << 1 << ' ' << a << ' ' << b << std::endl;
std::cin >> wynik;
```


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".

C++, wejście/wyjście przez `stdio`

Należy standardowo załączyć odpowiedni nagłówek (`#include <cstdio>`). Po wypisaniu każdego wiersza należy napisać `fflush(stdout)`. Przykładowe pytanie typu *niezależne* wygląda następująco:

```
printf("1 %d %d\n", a, b);
fflush(stdout);
scanf("%d", &wynik);
```

Python

Po wypisaniu każdego wiersza należy napisać `flush = True`. Przykładowe pytanie typu *niezależne* wygląda następująco:

```
print(f"1 {a} {b}", flush = True)
wynik = int(input())
```

Przykładowy przebieg programu

Załóżmy, że $n = 4$ oraz $m = 4$, kolejne drogi łączą pary miast $(0, 3), (0, 2), (1, 2), (2, 3)$, a ich koszty utrzymania to odpowiednio $3, 1, 2, 4$. Wówczas przebieg interakcji może wyglądać następująco:

Wywołana funkcja	**Wynik**	**Opis**
daj n	4	Otrzymujemy liczbę miast w Bajtocji.
daj drogi	$\{(0, 3), (0, 2), (1, 2), (2, 3)\}$	Otrzymujemy listę dróg w Bajtocji.
pytanie typu *gwiazda* z drogami $\{3, 1, 2\}$	1	Droga 1 ma najmniejszy koszt.
pytanie typu *niezależne* z drogami $0, 2$	1	Droga 0 ma większy koszt utrzymania niż droga 2.
pytanie typu *gwiazda* z drogami $\{3, 0\}$	0	Droga 0 ma mniejszy koszt utrzymania niż droga 3.
pytanie typu *niezależne* z drogami $2, 0$	-1	Droga 2 ma mniejszy koszt utrzymania niż droga 0.
zgłoszenie wyniku $\{0, 2, 1\}$	—	Udało nam się poprawnie wyznaczyć wynik.

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Dla każdego z nich program zostanie uruchomiony na pewnej liczbie testów. Ostateczny wynik w podzadaniu to minimum z wyników uzyskanych na wszystkich testach danego podzadania.

Podzadanie	**Ograniczenia**	**Punkty**
1	$n \leq 200$, $m \leq 300$	18
2	$n \leq 2000$, $m \leq 3000$	33
3	$m = n$	12
4	$m \leq n + 10$	16
5	brak dodatkowych ograniczeń	21

Wynik dla pojedynczego testu obliczany jest następująco: niech p będzie liczbą zapytań typu *niezależne*, a q liczbą zapytań typu *gwiazda*, które zadał Twój program. Wówczas wynik Twojego programu jest dany przez tabelę poniżej.

$p \backslash q$	$0 \leq q \leq n - 1$	$n - 1 < q \leq 2 \cdot (n - 1)$	$2 \cdot (n - 1) < q$
$0 \leq p \leq 15 \cdot m$	100%	75%	40%
$15 \cdot m < p \leq \max(15, n) \cdot m$	50%	30%	20%
$p > \max(15, n) \cdot m$	15%	10%	5%

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”.

Należy pamiętać, że wynik zostanie odpowiednio przeskalowany, jeśli program przekroczy połowę limitu czasu przewidzianego dla danego testu.

W przeciwnym wypadku, gdy udzielona odpowiedź jest błędna lub wystąpił dowolny inny błąd (błąd wykonania, przekroczenie limitu czasu itd.), wynik dla testu wynosi 0.

Testy przykładowe

Test 0 to test z przykładowego przebiegu powyżej. Poza tym:

`1ocen:` $n = 200$, $m = 199$. Nie można usunąć żadnej drogi.

`2ocen:` $n = 1500$, $m = 2998$. Wszystkie miasta poza jednym leżą na cyklu. Pozostałe miasto jest połączone drogą z każdym z miast na cyklu.

`3ocen:` $n = 200\,000$, $m = 300\,000$. Droga o numerze i łączy miasta i oraz $(i + 1) \bmod n$ dla $0 \leq i < n$. Droga o numerze $n + i$ łączy miasta i oraz $(i + 5) \bmod n$ dla $0 \leq i < m - n$. Koszt utrzymania drogi o numerze i wynosi $i + 1$ dla $0 \leq i < m$.

Dla zawodnika

W katalogu `dlazaw` znajdują się przykładowe (błędne) rozwiązania w C++ oraz Pythonie, które ilustrują obydwa modele komunikacji. Są też dostępne testy przykładowe (`red0.in`, `red[1-3]ocen.in`), biblioteki do komunikacji oraz przykładowy program sprawdzający. Testy przykładowe są w następującym formacie:

- w pierwszym wierszu: liczby całkowite n i m,
- w kolejnych m wierszach: po trzy liczby całkowite x_i, y_i, w_i oznaczające numery miast połączonych i-tą drogą oraz koszt jej utrzymania.

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 w SIO ani skrypt `ocen` na komputerach.

Opcja 1: Komunikacja przy pomocy biblioteki

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

- `g++ -O3 -static -std=c++20 redlib.cpp red.cpp -o red.e`

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

- C++: `./red.e < red0.in`
- Python: `python3 red.py < red0.in`

Opcja 2: Komunikacja przez standardowe wejście/wyjście

Pliki `red2.cpp` i `red2.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 `red2.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 `red0.in` komendami:

- C++: `./run.sh "./red2.e" red0.in`
- Python: `./run.sh "python3 red2.py" red0.in`