

Zadanie: TRA

Transport Publiczny



Potyczki Algorytmiczne 2026, runda próbna. Limity: 256 MB, 10 s.

10.03.2026

Bajtocja ma bardzo dobrze rozbudowaną siatkę komunikacyjną, składającą się z dwukierunkowych połączeń autobusowych i kolejowych. Każde połączenie (autobusowe lub kolejowe) łączy dwa spośród N Bajtockich miast.

Algosa zna dokładną siatkę połączeń autobusowych — dla i -tego z A połączeń wie jakie dwa miasta U_i i V_i ono łączy, a także liczbę C_i — koszt w bajtalarach przejechania tym połączeniem. Niestety, nie ma żadnych informacji o połączeniach kolejowych, więc będzie musiała współpracować z Bajtkiem. Bajtek co prawda nie wie o autobusach, ale dla każdego z B połączeń kolejowych wie, które dwa miasta ono łączy, oraz jaka jest cena za przejazd.

Algosa chciałaby dowiedzieć się, jaki jest najmniejszy koszt dojechania z jej miasta (miasto numer 1) do każdego z miast Bajtocji. W trasie Algosa może przesiadać się między autobusami i koleją (czyli przyjechać do jakiegoś miasta koleją i wyjechać autobusem lub na odwrót), może też kontynuować podróż tym samym środkiem transportu, którym przyjechała. Wiemy, że pomiędzy każdymi dwoma miastami istnieje przynajmniej jedna trasa (potencjalnie zawierająca zarówno autobusy jak i koleję).

Niestety, komunikacja jest droga. Algosa musi ustalić pożądane najmniejsze koszty wymieniając z Bajtkiem komunikaty składające się z jednego bitu (0 lub 1). Łączna liczba przekazanych (łącznie w obie strony) bitów nie może przekroczyć 58 000.

Interakcja

To jest zadanie interaktywne. Twój program zostanie uruchomiony w dwóch kopiach — jednej dla Algosi, a drugiej dla Bajtka. Każde z uruchomień na początku wejścia otrzyma słowo “Algosa” lub “Bajtek”, które określa za działanie której z osób odpowiada ta kopia programu.

W obu wypadkach, pierwsza linia wejścia zawiera tylko to jedno słowo. Druga linia wejścia zawiera dwie liczby. W wypadku Algosi jest to N ($1 \leq N \leq 2000$), czyli liczba miast w Bajtocji, oraz A ($0 \leq A \leq 500\,000$), liczba połączeń autobusowych. W wypadku Bajtka są to N (z tą samą wartością co w kopii Algosi), oraz B ($0 \leq B \leq 500\,000$), liczba połączeń kolejowych.

Kopia programu Algosi następnie zawiera A wierszy, określających siatkę połączeń autobusowych. W i -tej linii znajdują się trzy liczby U_i , V_i i C_i , ($1 \leq U_i, V_i \leq N$, $U_i \neq V_i$, $1 \leq C_i \leq 500$), oznaczające numery miast połączonych i -tym połączeniem, oraz koszt tego połączenia. Pomiedzy jedną parą miast istnieje co najwyżej jedno połączenie autobusowe.

Kopia programu Bajtka zawiera B wierszy, określającą siatkę połączeń kolejowych w tym samym formacie. Tu również pomiędzy jedną parą miast istnieje co najwyżej jedno połączenie kolejowe. Może natomiast się zdarzyć, że jakaś para miast jest połączona zarówno bezpośrednim połączeniem autobusowym, jak i kolejowym.

Następnie zaczyna się część interaktywna. Każda kopia może wypisać na wejście jedno z trzech poleceń (każde polecenie powinno być w oddzielnym wierszu):

- $+ c_i$, gdzie c_i to 0 albo 1 (czyli znak $+$, potem spacja, a potem 0 lub 1), oznaczające, że wysyłamy do drugiej osoby bit c_i .
- $?$, oznaczające, że chcemy przeczytać bit wysłany przez drugą stronę. Po wykonaniu tego polecenia, na wejściu znajdzie się jeden znak, odpowiadający pierwszemu nieprzeczytanemu bitowi wysłanemu przez drugą osobę.
- $!$, które to polecenie może wykonać tylko program Algosi. Po tym poleceniu program powinien wypisać w tym samym wierszu N liczb, gdzie i -ty liczba to odległość od miasta 1 do miasta i .

Po każdym wypisanym wierszu należy opróżnić bufor wyjścia, na przykład poprzez wywołanie polecenia `cout.flush()`, lub `fflush(stdout)` jeżeli używasz `printf`. Należy przestrzegać podanego wyżej formatu wejścia i wyjścia, łącznie z białymi znakami i podziałem na wiersze — w przeciwnym przypadku program może zakończyć się werdyktem *Przekroczenie limitu czasu* zamiast *Błędna odpowiedź*.

Przykładowa interakcja

Sprawdzarka do Algosi	Algosia do sprawdzarki	Sprawdzarka do Bajtka	Bajtek do sprawdzarki	Wyjaśnienie
Algosia		Bajtek		Sprawdzarka informuje obie kopie programu o tym, której z osób służą.
4 3		4 4		W Bajtocji są 4 miasta, 3 połączenia autobusowe, i 4 połączenia kolejowe.
1 2 6		2 3 3		Pierwsze połączenie autobusowe łączy miasta 0 i 1, kosztuje 6 bajtalarów. Pierwsze połączenie kolejowe łączy miasta 1 i 2 i kosztuje 3 bajtalary.
3 2 4		4 2 1		Opis połączenia 2 z 1 autobusem i 3 z 1 kolejną.
3 1 10		4 3 3		Zauważ, że do miasta 3 nie da się dojechać autobusem.
		4 1 7		Ostatnie połączenie kolejowe
	+ 1		?	Algosia wysyła bit 1 do Bajtka. Bajtek czeka na informację od Algosi.
	+ 0	1		Algosia wysyła bit 0 do Bajtka. Bajtek czyta uprzednio wysłany bit 1.
	?		+ 1	Algosia chce przeczytać bit wysłany przez Bajtka. Bajtek dopiero teraz (lub później) go wysyła.
1			?	Algosia otrzymuje wysłany przez Bajtka bit, Bajtek chce przeczytać drugi bit wysłany przez Algosię.
	! 0 6 9 7	0		Algosia już zna odpowiedź i ją zgłasza. Bajtek czyta drugi bit wysłany przez Algosię.

W tym przykładzie do miasta 1 nie trzeba w ogóle jechać, do miasta 2 można pojechać bezpośrednio autobusem za 6 bajtalarów, do miasta 4 można pojechać bezpośrednio pociągiem za 7 bajtalarów, natomiast do miasta 3 najtaniej dojechać jadąc wpierw autobusem do 2 za 6 bajtalarów, a potem przesiadając się na pociąg, który dowiezie nas do 3 za 3 bajtalary. Ta opcja jest tańsza niż bezpośrednie połączenie autobusowe z 1 do 3.

Eksperyment

W sekcji *Pliki* dostępny jest przykładowy interaktor `trasoc.cpp`. Może się on nieznacznie różnić od tego używanego do sprawdzania rozwiązań. Należy go uruchamiać komendą:

```
python3 trarun.py [rozwiązanie] < [test] > [wyjście]
```

przy czym plik `trasoc.cpp` musi być w tym samym folderze. Format, w którym interaktor przyjmuje wejście jest opisany w komentarzu w pliku `trasoc.cpp`.

Ocenianie

- W pierwszej grupie testów $A = 0$
- W drugiej grupie testów $B \leq 1000$
- W trzeciej grupie testów $A + B = N - 1$
- W czwartej grupie testów $N \leq 900$
- W piątej i szóstej grupie testów $N \leq 1100$
- W siódmej i ósmej grupie testów $N \leq 1400$