Bitocja

Dostępna pamięć: 32 MB.

W pewnym państwie zwanym Bitocją mieszka bardzo bogaty prezes BitBanku Bitazar. Codziennie dojeżdża on do pracy pokonując drogę z miasta 1 do miasta n. Do tej pory w Bitocji istniała sieć dwukierunkowych dróg pozwalających Bitazarowi dojechać do celu, lecz podróż w jego mniemaniu trwa zbyt długo. Prezes Bitazar ogłosił wśród firm budowlanych przetarg na budowę nowych połączeń, które pozwoliłyby mu zminimalizować czas dojazdu do pracy. W odpowiedzi otrzymał oferty. Dla każdej z nich musi rozstrzygnąć, czy dana droga skraca czas przejazdu z miasta 1 do miasta n. Jeśli tak, to firma buduje tę drogę, a Bitazar rozważa kolejne propozycje przyjmując, że droga została wybudowana. W przeciwnym wypadku rozważana jest następna oferta, a stan dróg nie ulega zmianie. Twoim zadaniem jest pomóc prezesowi w wyborze nowych dróg do budowy.

Zadanie

Opracuj program, który:

- wczyta ze standardowego wejścia opis dróg istniejących oraz propozycji nowych połączeń,
- dla każdej proponowanej nowej drogi odpowie na pytanie, czy spełnia ona wymagania prezesa Bitazara,
- wypisze na standardowe wyjście wynik.

Wejście

Pierwszy wiersz zawiera trzy liczby: n ($1 \leq n \leq 100$), k ($1 \leq k \leq \frac{n(n-1)}{2}$) i n ($1 \leq m \leq 10.000$), czyli kolejno ilość miast (miasta są ponumerowane liczbami całkowitymi z zakresu $[1;n]$), ilość dróg już wybudowanych oraz ilość propozycji nowych dróg do wybudowania. Kolejne k wierszy zawiera opis dróg już istniejących, a dalsze m wierszy opis propozycji nowych dróg. Opis drogi już istniejącej, jak i propozycja składa się z trójki liczb (a, b, w), gdzie a i b to numery miast, które łączy dana droga ($1 \leq a,b \leq n$), oraz w — czas przejazdu daną drogą $1 \leq w \leq 1.000.000$).

Wyjście

Dla każdej propozycji nowej drogi wypisz 1, jeśli Bitazar powinien ofertę przyjąć, albo 0 jeśli powinien ją odrzucić.

Przykład

Dla danych wejściowych:

```
4 5 5
1 4 7
1 2 2
2 4 7
```

```
3 4 2
1 3 6
1 3 5
1 4 6
2 4 4
1 3 3
2 4 2
```

poprawnym wynikiem jest:

```
0
1
0
1
1
```

Rozwiązanie

Sprowadźmy problem do języka teorii grafów. Dany jest graf nieskierowany o krawędziach ważonych, w którym poszukujemy długości najkrótszej ścieżki z wierzchołka o numerze 1 do wierzchołka n-tego. Ponadto możemy do grafu dodawać nowe krawędzie, które mogą poprawić tę wartość. Nasze zadanie polega na sprawdzeniu które z tych krawędzi istotnie polepszą najkrótszą ścieżkę $1 \rightsquigarrow n$. W sytuacji takiej poprawy dorzucamy tę krawędź do grafu przed sprawdzeniem następnej propozycji nowej drogi.

Ponieważ dla każdej potencjalnie nowej drogi sprawdzamy, czy najkrótsza ścieżka z 1 do n ulegnie zmianie, rozpatrzmy kilka algorytmów obliczających długości najkrótszych ścieżek. Do najpopularniejszych należą algorytmy Dijkstry, Forda–Bellmana i Floyda–Warshalla. W naszym grafie mamy zawsze krawędzie o nieujemnych wagach, możemy więc usunąć z rozważań algorytm Forda–Bellmana, który byłby dość nieefektywny w porównaniu do algorytmu Dijkstry (złożoność $O(nk)$).

Równie nieprzydatny wydaje się algorytm Floyda–Warshalla. Co prawda złożoność obliczeniowa $O(n^3)$ wydaje się satysfakcjonująca dla maksymalnej ilości wierzchołków, jednak należy pamiętać, że dla każdej nowej krawędzi należy przeprowadzić obliczenia na nowo, co w końcowym algorytmie dałoby czas działania rzędu $O(n^3m)$ — zdecydowanie zbyt wiele.

Wróćmy zatem do algorytmu Dijkstry, który w najprostszej postaci działa w czasie $O(n^2+k)$. W rzeczywistości może okazać się to odrobinę zbyt powolne na grafach gęstych, czyli takich, w których współczynnik k będzie wysoki. Nawet zmodyfikowany algorytm Dijkstry wykorzystujący strukturę kopców binarnych jako kolejkę priorytetową nie pomoże, a nawet przeciwnie. Złożoność teoretycznie spadnie do wartości $O((n+k)log\ n)$, jednak ponownie rozważmy grafy bardzo gęste, w szczególności grafy pełne (każda para wierzchołków jest połączona krawędzią), gdzie współczynnik k osiąga rząd wielkości $O(n^2)$. W takiej sytuacji otrzymany algorytm może okazać się wolniejszy od poprzedniego! Co więcej, z każdą dodaną nową drogą rośnie współczynnik k.

Problemem algorytmu Dijkstry jest jego zależność od ilości krawędzi w grafie i tak naprawdę trudno cokolwiek konstruktywnego z tym zrobić. Jedyny rozważany algorytm niezależny od k odrzuciliśmy jako zbyt wolny przy wielokrotnych iteracjach — do zagwarantowania poprawności przeliczamy najkrótsze ścieżki pomiędzy każdą parą wierzchołków. Tak się składa, że można wykorzystać specyficzną konstrukcję algorytmu Floyda–Warshalla do jego ograniczenia bez straty interesujących nas własności. Lecz najpierw kluczowy…

Fakt. *Dodanie krawędzi $u \leftrightarrow v$ może wpłynąć tylko na długość ścieżek korzystających z tej krawędzi.*

Przypomnijmy działanie algorytmu Floyda–Warshalla. Dla każdej pary wierzchołków u, v szukamy najkrótszej ścieżki między nimi. W tym celu sprawdzamy, czy dla dowolnego innego wierzchołka w zachodzi

$$|u \rightsquigarrow v| > |u \rightsquigarrow w| + |w \rightsquigarrow v|$$

Jeśli tak, to znaczy, że możemy poprawić najkrótszą ścieżkę z u do v na wartość po prawej stronie nierówności. W rzeczywistości powyższy warunek oznacza sytuację, w której znaleźliśmy jakąś ścieżkę $u \rightsquigarrow v$, ale wkrótce okazuje się, że krótsza trasa wiedzie przez wierzchołek pośredni w.

Ponieważ sprawdzamy tylko ścieżki korzystające z nowo dodanej krawędzi (oznaczmy ją $x \leftrightarrow y$), zmodyfikowany algorytm Floyda–Warshalla będzie wyglądał tak: dla każdej pary wierzchołków u, v szukamy najkrótszej ścieżki między nimi. W tym celu sprawdzamy, czy ścieżka $u \rightsquigarrow x \leftrightarrow y \rightsquigarrow v$ lub $u \rightsquigarrow y \leftrightarrow x \rightsquigarrow v$ jest krótsza od ścieżki $u \rightsquigarrow v$.

Wniosek. *Aktualizację najkrótszych ścieżek po dodaniu nowej krawędzi możemy wykonać w czasie* $O(n^2)$.

Jeden raz obliczamy "konwencjonalnie" odległości pomiędzy każdą parą wierzchołków, a następnie dla każdej nowej krawędzi sprawdzamy, czy po jej dodaniu uzyskamy korzystniejszy wynik dla $1 \rightsquigarrow n$. Jeśli tak, dodajemy tę krawędź do grafu i aktualizujemy odległości. Ostateczna złożoność wynosi więc $O(n^3 + n^2 m)$, co przy niskim koszcie obliczeniowym podstawowej operacji algorytmu Floyda–Warshalla jest wartością akceptowalną.