Opracowanie: j/
Zadanie J: Trasowanie

HISTORIA:
- wersja 1.1: 2007, Marcin Michalski
- wersja 1.0: 2007, Tomasz Idziaszek

dokument systemu SINOL 1.3.1

1 Analiza problemu

Wysłówmy nasz problem w języku teorii grafów. Jest dany **skierowany graf** $G = (V, E)$, mający n wierzchołków. W G wyróżnimy dwa specjalne wierzchołki: początkowy s i końcowy t. Chcemy znaleźć dwie **skierowane ścieżki** w tym grafie: jedną z s do t i drugą w przeciwnym kierunku. Innymi słowy potrzebujemy ścieżek $v_0 \to v_1 \to \cdots \to v_{k-1} \to v_k$ i $\bar{v}_0 \to \bar{v}_1 \to \cdots \to \bar{v}_{l-1} \to \bar{v}_l$ takich, że $v_0 = \bar{v}_l = s$, $v_k = \bar{v}_0 = t$. Jesteśmy zainteresowani takimi ścieżkami, które zawierają minimalną liczbę wierzchołków, to znaczy chcemy zminimalizować moc zbioru $\{v_0, \ldots, v_k, \bar{v}_0, \ldots, \bar{v}_l\}$.

2 Rozwiązanie

Na początkek wprowadźmy **funkcję odległości** $\delta: V \times V \to \mathbb{N}$. Dla każdej pary wierzchołków $u, v \in V$ kładziemy $\delta(u, v) = \infty$ jeżeli nie istnieje żadna skierowana ścieżka w G z u do v. W przeciwnym wypadku $\delta(u, v) = k$ jeżeli takie ścieżki istnieją i najkrótsza z nich ma długość k (zawiera k krawędzi).

Dla zdania logicznego P nawias Iversona $[P]$ jest zdefiniowany następująco: $[P] = 1$ jeżeli P jest prawdziwe, $[P] = 0$ w przeciwnym wypadku.

Zdefiniujemy drugi graf skierowany $G_2 = (V \times V, E_2)$ oraz **funkcję wagową** $w: E_2 \to \mathbb{N}$ jak następuje:

- dla każdej krawędzi $(u, v) \in E$ i każdego wierzchołka $v_0 \in V$ dodajemy do E_2 **krawędź pierwszego rodzaju** $e = ((u, v_0), (v, v_0))$ i kładziemy $w(e) = [v \neq v_0]$;
- dla każdej krawędzi $(v, u) \in E$ (zauważmy, że u i v są tu zamienione) i każdego wierzchołka $v_0 \in V$ dodajemy do E_2 **krawędź drugiego rodzaju** $e = ((v_0, u), (v_0, v))$ i kładziemy $w(e) = [v \neq v_0]$;
- dla każdej pary różnych wierzchołków $u, v \in V$ jeżeli $\delta(u, v) \neq \infty$ dodajemy **krawędź trzeciego rodzaju** $e = ((u, v), (v, u))$ do E_2 i definiujemy $w(e) = \delta(u, v) - 1$.

Szukanym rozwiązaniem jest długość najkrótszej ścieżki w G_2 z (s, s) do (t, t) przy użyciu funkcji wagowej w.

3 Dowód poprawności

Powiedzmy, że mamy dwie ścieżki (jedną z s do t i drugą z t do s) które są rozwiazaniem naszego problemu. Niech $a_1, a_2, \ldots$ będzie zbiorem wierzchołkow wspólnych dla obydwóch ścieżek w kolejności w jakiej te wierzchołki występuja na pierwszej ścieżce.

Jeżeli a jest zbiorem pustym (czyli ścieżki nie mają wspólnych wierzchołków poza początkiem i końcem) wtedy łatwo jest dowieść, że nasze rozwiązanie jest równe najkrótszej scieżce od (s, s) do (t, t) w grafie G_2. Wystarczy użyc krawędzi pierwszego rodzaju aby przejśc od (s, s) do (t, s), których koszt jest równy długości pierwszej ścieżki (od s do t) i poźniej przejśc od (t, s) do (t, t) używając krawędzi drugiego typu.

Jeżeli a jest niepuste, to mamy dwie możliwosci: albo a_1 występuje jako ostatni wierzchołek ze zbioru a na ściezce od t do s, albo istnieje takie k, że a_k występuje po a_1, ale a_{k+1} już nie.

W pierwszym przypadku możemy podzielic nasz problem na podproblemy, dla których wartości s, t będą równe odpowiednio s, a_1 i a_1, t. Istnieje rozwiazanie pierwszego podproblemu, które nie ma punktów wspólnych (wystarczy wziaśc nasze rozwiazanie odpowiednio odcięte), tak więc dla tego przypadku udowodnilismy, ze jest on równoważny odpowiedniej najkrótszej ścieżce w G_2. Dla drugiego podproblemu możemy rekurencyjnie użyc naszego dowodu.

W drugim przypadku będziemy musieli zacząc używac krawędzi trzeciego rodzaju. Na początku trzeba zauważyc, że istnieją krawędzie $(a_1, a_2), (a_2, a_3), \ldots (a_{k-1}, a_k)$ i że należą one do obydwóch ścieżek. Co więcej na drugiej ścieżce pomiędzy wierzchołkami a_{k+1} a a_1 nie występuje żaden wierzchołek ze zbioru a. Tak więc znowu mozemy podzielic nasz problem na dwa podproblemy, gdzie wartościami s, t będą odpowiednio s, a_{k+1} i a_{k+1}, t.

Teraz pozostało nam dowieśc, że istnieje ścieżka w G_2 pomiedzy (s, s) a (a_{k+1}, a_{k+1}) o długości nie wiekszej niż rozwiazanie pierwszego podproblemu. Jeżeli weźmiemy ścieżke w G_2 od (s, s) do (a_1, s) używajac krawędzi pierwszego rodzaju, później do (a_1, a_k) po krawędziach drugiego rodzaju. Następnie używamy krawędzi trzeciego rodzaju z (a_1, a_k) do (a_k, a_1) i później znowu pierwszego rodzaju do (a_{k+1}, a_1) i na końcu drugiego rodzaju dochodząc do (a_{k+1}, a_{k+1}).

Krawędzie pierwszego i drugiego rodzaju odpowiadają przesuwaniu się pierwszą ścieżką w przód i drugą ścieżka w tył. Dodatkowo krawędź trzeciego rodzaju odpowiada przesunięciu się obiema ścieżkami po tych samych krawędziach. Tak więc jeżeli istnieje w G_2 ścieżka od (a, b) do (c, d) o koszcie x to oznacza, że istnieją dwie ścieżki w G, pierwsza od a do c i druga od d do b takie, że sumaryczna ilośc wierzchołkow przez jakie przechodzą jest równa x.

Tak więc dowiedliśmy, że dla kazdej ścieżki w G_2 isnieją odpowiadające jej ścieżki w G. Dodatkowo wiemy, ze długośc najkrotszej ścieżki w G_2 jest nie większa niż rozwiazanie dla G. Stąd wynika już, ze długośc najkrótszej ścieżki między odpowiednimi wierzchołkami w G_2 jest równa odpowiedzi na nasz problem w G.

4 Algorytm

Po wczytaniu grafu G z wejścia, konstruujemy go w pamięci komputera razem z jego dopełnieniem G^T. Następnie obliczamy funkcję δ dla każdej pary wierzchołków z G. Możemy to zrobić używając **algorytmu Floyda-Warshalla** działającego w czasie $O(n^3)$.

Następnym krokiem jest wykonanie **algorytmu Dijkstry** na grafie G_2 z funkcją wagową w. Zauważmy, że będziemy na bieżąco konstruować graf G_2. Za każdym razem, gdy z kolejki priorytowej wyciągamy wierzchołek (u_1, u_2) przechodzimy po liście sąsiedztwa u_1 i uaktualniamy krawędzie pierwszego rodzaju, następnie przechodzimy po liście sąsiedztwa u_2 w grafie transponowanym G^T uaktualniając krawędzie drugiego rodzaju; w końcu sprawdzamy czy $u_1 \neq u_2$ i jeśli tak, uaktualniamy krawędź trzeciego rodzaju.

Graf G_2 ma $O(n^2)$ wierzchołków. Każdy z nich jest początkiem $O(n)$ krawędzi pierwszego i drugiego rodzaju i co najwyżej jednej krawędzi trzeciego rodzaju. Zatem mamy $O(n^3)$ krawędzi w G_2. Jeżeli zaimplementujemy kolejkę priorytetową w algorytmie Dijkstry za pomocą kopca, całkowity czas algorytmu będzie wynosił $O(n^3 \log n)$.

Powyższy algorytm został zaimplementowany w C++ i w Javie (programy `j.cpp` i `j.java`).

5 Niepoprawne rozwiązania

Implementując kolejkę priorytetową w algorytmie Dijkstry za pomocą tablicy, uzyskujemy czas $O(n^4)$, co daje algorytm zbyt wolny.

Typowym błędem jest implementacja powyższego algorytmu bez uwzględniania krawędzi trzeciego rodzaju.

Również algorytmy, które ustalają najkrótszą ścieżkę w G, a potem próbują dobrać do niej drugą, skazane są na niepowodzenie.

6 Testy

Pierwsze kilka testów, to małe testy poprawnościowe.

Test nr 2 to prosta pętla.

Test 3 sprawdza, czy program uwzględnia krawędzie 3 typu.

Test 4 równiez używa krawedzi 3 typu i dodatkowo wyłapuje algorytmy próbujace znajdowac najkrótsze ścieżki.

Test 5 jest testem z prawie pełnym grafem o wielkości 50.