Efektywne algorytmy znajdowania maksymalnego przepływu notatki do ćwiczeń (z o. o.)

Łukasz Kowalik
kowalik@mimuw.edu.pl

17 kwietnia 2002

1 Oznaczenia

Oznaczenia przyjmiemy według książki Cormena, Leisersona i Rivesta „Wprowadzenie do algorytmów", w szczególności:

- Dla dowolnych u, v przez $c(u, v) \geqslant 0$ oznaczamy przepustowość (gdy $u - v \notin E$ to c(u,v)=0).
- $f(u, v)$ oznacza przepływ po krawędzi od u do v.
- $c_f(u, v) = c(u, v) - f(u, v)$ jest przepustowością residualną, a sieć indukowana przez niezerowe przepustwowści residualne oznaczamy przez G_f i nazywamy siecią residualną.
- *Ścieżka powiększająca* jest dowolną ścieżką od s do t w sieci G_f.
- Krawędź $u \to v$ jest *nasycona*, gdy $f(u, v) = c(u, v)$.

2 Algorytm Dinica (1970)

2.1 Graf warstwowy

Niech G będzie siecią o źrodle s i ujściu t. Niech f będzie przepływem w G. Zdefiniujmy *graf warstwowy* L_f jako podgraf grafu G_f taki, że $u \to v \in E(L_f)$ wtedy i tylko wtedy gdy $d_{G_f}(s, v) = d_{G_f}(s, u) + 1$. Innymi słowy krawędź $e = u \to v \in E(L_f)$ wtedy i tylko wtedy gdy e zawiera się w pewnej najkrótszej ścieżce od s do v. Zauważmy, że L_f zawiera wszystkie najkrótsze ścieżki powiększające. Uwaga ! L_f nie jest drzewem BFS !

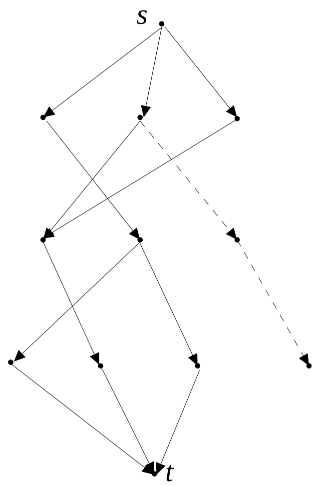
Graf L_f łatwo utworzyć w czasie $O(n + m)$ korzystając z BFS-u:

1. zapuszczamy BFS z s i obliczamy odległości wszystkich wierzchołków od s
2. dla każdej krawędzi $u \to v$ sprawdzamy, czy $d_{G_f}(s, v) = d_{G_f}(s, u) + 1$ i jeśli tak, dodajemy ją do grafu.

2.2 Schemat algorytmu Dinica

Przepływem blokującym w L_f nazywamy dowolny przepływ w L_f taki, że każda ścieżka od s do t w L_f zawiera krawędź nasyconą. Teraz możemy już podać schemat algorytmu Dinica:

$f := 0$;
dopóki w L_f istnieje ścieżka od s do t
    znajdź przepływ blokujący f^* w L_f
    $f := f + f^*$



Rysunek 1: Graf warstwowy L_f. Krawędzie przerywane zostaną usunięte przez procedurę OCZYSC.

Możemy powiedzieć, że algorytm Dinica składa się z *faz*, a każda faza polega na skonstruowaniu grafu L_f i znalezieniu w nim przepływu blokującego. Poprawność algorytmu Dinica wynika natychmiast z twierdzenia o maksymalnym przepływie i minimalnym przekroju (po zakończeniu algorytmu nie ma już ścieżek powiększających).

2.3 Znajdowanie przepływu blokującego w grafie warstwowym L_f

Załóżmy, że procedura OCZYŚĆ usuwa z grafu L_f wszystkie wierzchołki u takie, że $u \neq t$ oraz nie ma krawędzi wychodzących z u. Zauważmy, że „oczyszczony" graf L_f jest dokładnie grafem składającym się z najkrótszych ścieżek powiększjących. W takim grafie możemy znaleźć ściężkę powiększającą w czasie $O(n)$. Następujący algorytm znajduje przepływ blokujący w L_f:

$f^* := 0$;
while $L_f \neq \emptyset$ **do**
    OCZYSC;
    Znajdź ścieżkę p od s do t (w czasie $O(n)$)
    $\Delta :=$ najmniejsza przepustowość na p
    Po p przesyłamy Δ jednostek przepływu, powiększając $|f^*|$ o Δ.
    Usuwamy z p wszystkie krawędzie nasycone (takie, że $f^*(u,v) = c_f(u,v)$)

Poprawność powyższego algorytmu jest oczywista: niech p będzie dowolną ścieżką od s do t w L_f. Algorytm podczas swojego działania usunie co najmniej jedną krawędź z p. Widać, że pierwsza usunięta krawędź będzie nasycona.

Zajmijmy się teraz złożonością czasową algorytmu znajdowania przepływu blokującego. Na początek zauważmy, że można ten algorytm zaimplementować tak, aby całkowita liczba operacji wykontwanych przez procudurę OCZYŚĆ była $O(n+m)$. Wystarczy w tym celu dla każdego wierzchołka pamiętać liczbę wychodzących z niego krawędzi. Ponieważ pozostałe opeacje wewnątrz pętli zajmują czas $O(n)$, a pętla wykonuje się $O(m)$ razy (podczas każdego obrotu pętli co najmniej jedna krawędź staje się nasycona), więc całkowity czas wynosi $O(nm)$.

2.4 Złożoność algorytmu Dinica

Pozostało nam jedynie oszacować liczbę faz wykonywanych przez algorytm Dinica. Skorzystamy tu z faktu dowodzonego na wykładzie na potrzeby algorytmu Edmondsa-Karpa (dowód można też znaleźć w Cormenie). Niech $\delta_f(u,v)$ oznacza odległość od u do v w sieci residualnej G_f.

Lemat 1 (z analizy algorytmu E-K). *Po każdym powiększeniu przepływu wzdłuż ścieżki powiększającej, dla dowolnego wierzchołka $v \in V - \{s,t\}$ odległość $\delta_f(s,v)$ nie maleje.*

Udowodnimy następujący fakt:

Lemat 2. *Po każdej kolejnej fazie algorytmu Dinica długość najkrótszej ścieżki powiększającej rośnie.*

Dowód. Załóżmy przeciwnie, że po zakończeniu pewnej fazy algorytmu Dinica długość najkrótszej ścieżki w G_f nie wzrosła. Niech p będzie taką ścieżką oraz niech f oznacza przepływ obliczony przed rozpoczęciem tej fazy, a f' przepływ po jej zakończeniu. Ścieżka p nie mogła cała zawierać się w grafie L_f, czyli istnieje krawędź $u \to v \in p$ taka, że $u \to v \notin E(L_f)$. Niech $u \to v$ będzie ostatnią taką krawędzią na ścieżce p. Taka krawędź mogła „powstać” tylko w ten sposób, że pewna liczba jednostek przepływu została przepchnięta od v do u, czyli $v \to u \in E(L_f)$ oraz $v \neq t$. Ale wtedy $\delta_f(s,u)$ nie zmalała (z lematu 1), czyli $\delta_f(s,v)$ wzrosła o 2. Wtedy $\delta_{f'}(s,t) > \delta_f(s,t)$, czyli dlugosc najkrotszej sciezki musiala wzrosnac. □

Wniosek 1. *Liczba faz w algorytmie Dinica wynosi co najwyżej $n-1$.*

Wniosek 2. *Algorytm Dinica ma złożoność czasową $O(n^2m)$*

3 Algorytm trzech Hindusów (1978)

Nazwiska tytułowych trzech Hindusów to Malhotra, Kumar, Maheshwari. Dlatego ten algorytm jest też czasem nazywany algorytmem MKM. Schemat algorytmu trzech Hindusów jest taki sam jak dla algorytmu Dinica. Różnica polega na innym podejściu do znajdowania przepływu blokującego w grafie warstwowym L_f.

Kluczowym pojęciem jest tu *przepustowość wierzchołka* w sieci (G,c), którą definiujemy jako

$$c(v) = min\{\sum_{u\in V} c(u,v), \sum_{u \in V} c(v,u)\}$$

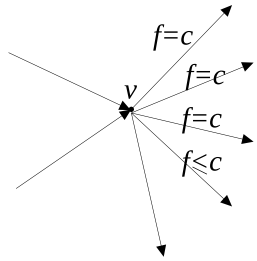
Oczywiście rozważając przepustowość wierzcholków w sieci L_f, będziemy korzystali z przepustowości residualnych c_f.

3.1 Algorytm znajdowania przepływu blokującego w L_f

Algorytm znajdowania przepływu blokującego w L_f sprowadza się do powtarzania, dopóki w grafie L_f zostały jeszcze jakieś wierzchołki różne od s i t, następującego ciągu operacji:

1. Znajdź wierzchołek $v \notin \{s,t\}$ o najmniejszej przepustowości d. Jeśli $d = 0$ skocz do 3.
2. Prześlij d jednostek przepływu z v do t i cofnij d jednostek przepływu z v do s.
3. Usuń v i wszystkie incycentne z nim krawędzie. Popraw przepustowość sąsiednich wierzchołków.

Krótkiego komentarza wymaga jeszcze krok (2). d jednostek przepływu przesyłamy od v do t zachłannie; jednostki przepływu przepychamy do kolejnych warstw grafu L_f, rozładowując wierzchołki z kolejnych warstw. Robimy to w ten sposób, że po każdej krawędzi przesyłamy *największą możliwą* liczbe jednostek przepływu. Innymi słowy, rozładowując wierzchołek x, nasycamy pewną liczbę krawędzi wychodzących z x, co najwyżej jedną krawędź (spośród krawędzi, po których przesyłamy przepływ) pozostawiając nienasyconą, jak na rysunku 2. Zupełnie analogicznie cofamy przepływ z v do s. Zauważmy, że ponieważ wierzchołek v miał najmniejszą przepustowość, opercja (2) zawsze się powiedzie.



Rysunek 2: Krok (3): zachłanne rozładowywanie wierzchołka v. Pewna liczba krawędzi zostaje nasycona, co najwyżej jedna użyta krawędź pozostaje nienasycona, pozostałych krawędzi nie używamy

3.2 Analiza złożoności czasowej

Dokładnie tak samo jak w algorytmie Dinica liczba faz nie może przekroczyć $n-1$. Wystarczy więc oszacować czas potrzebny na wykonanie jednej fazy.

Znajdowanie przepływu blokującego może zająć co najwyżej $O(n)$ obrotów pętli, ponieważ za każdym razem usuwany jest co najmniej jeden wierzchołek. Obliczmy łączną liczbę operacji potrzebnych do wykonania kroku (2). Przesłanie jednostek przepływu po pojedynczej krawędzi zajmuje czas stały. Co najwyżej m razy przesyłamy jednostki przepływu nasycając krawędzie. Z drugiej strony, łączna liczba operacji przesłania przepływu, które nie nasycają krawędzi nie może przekroczyć n^2 (n podczas jednego obrotu pętli, ponieważ dla każdego rozładowywanego wierzchołka wykonujemy co najwyżej jedną taką operację). Stąd krok (2) zajmuje łącznie czas $O(m+n^2) = O(n^2)$.

Łatwo widać, że krok (1) zajmuje czas $O(n)$, czyli łącznie $O(n^2)$, natomiast łączna liczba operacji wykonywanych w kroku (3) nie przekracza $O(n+m)$. Możemy więc odnotować fakt:

Fakt 1. Znajdowanie przepływu blokującego w sieci L_f zgodnie z algorytmem trzech Hindusów zajmuje czas $O(n^2)$

Wniosek 3. *Algorytm trzech Hindusów ma złożoność czasową $O(n^3)$.*

4 Literatura

Algorytm trzech Hindusów jest opisany w książce Lipskiego *kombinatoryka dla programistów*. Algorytm Dinica i jego analizę można znaleźć w internecie. Algorytmy Forda-Fulkersona i Edmondsa-Karpa wraz z dokładną analizą są w Cormenie (*Wprowadzenie do algorytmów*).