Algorytmika@mimuw

Stanisław Morawski na podstawie wykładów na MIM UW

9 marca 2018

Spis treści

0.1 Zasady zaliczania

Podczas semestru będą do zrobienia 2 pracy domowe. Żeby być dopuszczonym do egzaminu trzeba mieć $\geqslant 50\%$. Nie ma skryptu, są materiały na stronie Kowalika.

Niniejszym oddaje skrypto-notatki w Wasze ręce.

Rozdział 1

Przepływy

Definicja 1 (Sieć przepływowa). *Siecią przepływową nazywamy N = (G,c,s,t), gdzie:*

1. $G = (V, E)$
2. $c = E \to \mathbb{R}_+$ *przy czym możemy pisać jakby było* $V \times V \to \mathbb{R}_+$
3. $s, t \in V$

Definicja 2 (Przepływ). *Przepływem w sieci przepływowej N nazywamy funkcję* $f : V \times V \to R$ *taką, że:*

1. $\forall_{(u,v)\in E} |f(u,v)| \leqslant c(u,v)$
2. $\forall_{(u,v)\in E} f(u,v) = -f(u,v)$
3. $\forall_{(v)\in E\setminus\{s,t\}} \sum_{u\in E} f(u,v) = 0$

Definicja 3 (Warotść przepływu). *Wartością przepływu nazywamy* $|f| = \sum_v f(s,v)$

Będziemy analizowali algorytmy szukamjące przepływu o największej wartości.

Uwaga: Od tej pory zakładamy że $\exists_{(u,v)\in E} \Rightarrow !\exists_{(v,u)\in E}$.
Jeśli jest inaczej to zawsze możemy stworzyć nowy wierzchołek (uv) i krawędź $u \to v$ zastąpić krawędziami $u \to (uv) \to v$ z odpowiednimi wagami. Równoważność tych dwóch sieci przepływowych jest oczywista.

1.1 Algorytm zachłanny - Forda-Fulkersona

Szkic:

$f := 0$ $\quad\quad (\iff \forall_{u,v} f(u,v) = 0)$
for all $i \in E$ **do**
  powiększaj f wzdłuż ścieżek od s do t
end for

Definicja 4 (Sieć residualna). *Sieć residualna dla przepływu f w N to sieć* $N_f = (G = \{V, E_f\}, c_f, s, t)$*, gdzie:*

- $c_f(u,v) = c(u,v) - f(u,v)$ *(przepustowość residualna)*
- $E_f = \{(u,v) \in V^2 : c_f(u,c) > 0\}$

Definicja 5 (Ścieżka powiększająca). *Ścieżka powiększająca to dowolna ścieżka* $s \to t \in G_f$

Niech będą dane przepływ f, ścieżka powiększająca P.

Niech $f_p : V^2 \to R$

$$f_p(u,v) = \begin{cases} min_{(x,y)\in P} c_f(x,y) & \text{gdy}(u,v) \in P \\ min_{(x,y)\in P} c_f(x,y) & \text{gdy}(v,u) \in P \\ 0 \quad \text{wpp} \end{cases}$$

Note: f_p jest przepływem w N_f

Będziemy używać dodawania funkcji: $(f + f_p)(x) = f(x) + f_p(x)$

Algorytm FF:
  $f \leftarrow 0$
  while $\exists P$ **do**
    $f := f + f_p$
    uaktualnij N_f
  end while

Czasy:

- istnienie P sprawdzamy w O(G_f) za pomocą DFS-a
- uaktualnianie N_f w O(P)

$|f^*|$ - maksymalny przepływ

Jeśli przepustowości są całkowite $\iff c : V^2 \to \mathbb{N}$, wtedy mamy $\leqslant |f^*|$ iteracji (bo w każdej iteracji $|f|$ się zwiększa), zatem czas $\leqslant O(|f^*|E)$

1.1.1 Dowód poprawności

Definicja 6 (Przekrój)**.** *Przekrojem w sieci $N = (G, c, s, t)$ nazywamy dowolny podział V na zbiory S, T takie że* $S \cup T = V \wedge S \cap T = \emptyset$

Definicja 7 (Przepustowość przekroju i przepływ przez przekrój)**.** *Przepustowością przekroju nazywamy* $c(S,T) = \sum_{u\in S, v\in T} c(u,v)$*, zaś przepływem przez przekrój* $f(S,T) = \sum_{u\in S,v\in T} f(u,v)$

Lemat 1. *f przepływ, P ścieżka powiększająca $\Rightarrow f + f_p$ jest przepływem*

Lemat 2. *f przepływ w N, g przepływ w $N_f \Rightarrow (f + g)$ jest przepływem w N*

Lemat 3. $f(S,T) \leqslant c(S,T)$ *dowód oczywisty na podstawie definicji.*

Lemat 4. $\forall_{(S,T)} f(S,T) = |f| = f(\{s\}, V \setminus \{s\}) = \sum_{v\in V} f(s,v)$

Dowód. $f(S,T) = \sum_{u\in S,v\in T} f(u,v) = \sum_{u\in S,v\in T} f(u,v) - \sum_{u\in S,v\in S} f(u,v)$ wiemy, że $\sum_{u\in S,v\in S} f(u,v) = 0$ zatem to dalej jest równe $= \sum_{v\in V} f(s,v) + \sum_{u\in S\setminus\{s\},v\in V} f(u,v)$ przy czym jak drugi składnik rozbijemy na $\sum\sum$ to z warunku (3) z definicji przepływu = 0. zatem to dalej jest równe $|f|$ □

Po co nam to? Jak połączymy te dwie nierówności to mamy: $\forall_f \forall_{S,T} |f| \leqslant c(S,T)$

Chcąc udowodnić że algorytm tworzy maksymalny przepływ pokażemy że powyżej zajdzie równość.

Twierdzenie o maksymalnym przepływie i minimalnym przekroju - jeśli f to przepływ w N to są równoważne:

1. f jest maksymalny
2. nie istnieje ścieżka powiększająca
3. istnieje przekrót (S,T) taki, że $c(S,T) = |f|$

Dowód:

- $1 \Rightarrow 2$ bo gdyby $\exists P \Rightarrow f + f_p > f$

- $3 \Rightarrow 1$ z czegoś wyżej
- $2 \Rightarrow 3$ Buduję przekrój $S :=$ wierzchołki osiągalne z $swG_f.T := V \setminus S$. To jest przekrój. $|f| = \sum_{u \in S, v \in T} f(u,v) = \sum_{u \in S, v \in T} c(u,v)$. Ostatnia równość zachodzi bo 2.

Zatem jeśli algorytm się zatrzyma, to nie istnieje ścieżka powiększająca, zatem wyznacza maksymalny przepływ. Działa też dla wymiernych, bo można pomnożyć razy wszystkie mianowniki.

1.2 Algorytm Edmondsa-Karpa

Spróbujemy wybrać jakoś ścieżkę powiększającą w algorytmie Ford-Fulkersona, żeby to optymalizować.

Pierwszy pomysł: weźmy najkrótsze ścieżki (w sensie liczby krawędzi). Najkrótszą ścieżkę będziemy wyszukiwać BFS-em.

Poprawność jest oczywista na podstawie poprawności Forda-Fulkersona.

Zastanówmy się nad złożonością.

Iteracja dalej trwa $O(E)$. Spróbujemy znaleźć ograniczenie na liczbę iteracji. Będziemy pisać odległość $s \to v$ w G_f jako $\delta_f(s,v)$ Zaczniemy od pokazania:

Lemat 5. *f - przepływ, P - najkrótsza ścieżka powiększająca, $f' = f + f_p \Rightarrow \forall_{v \in V} \delta_{f'}(s,v) \geqslant \delta_f(s,v)$*

Dowód. Indukcja:

jeśli $(w,v) \in E_f$ to $\delta_{f'}(v) = \delta_{f'}(w) + 1 \geqslant \delta_f(w) + 1 \geqslant \delta_f(v)$

$(w,v) \notin E_f \Rightarrow (v,w) \in E_f \Rightarrow \delta_{f'}(w) = \delta_f(v) + 1 \Rightarrow \delta_{f'}(v) \geqslant \delta_f(w) + 1 = \delta_f(v) + 2$ □

Lemat 6. *W algorytmie Edmondsa-Karpa każda krawędź staje się nasycona $O(V)$ razy.*

Z tego jasno wyniknie że liczba iteracji jest $O(V * E)$ (przy każdej iteracji conajmniej jedna krawędź staje się nasycona). (Dopiero teraz pokazaliśmy, że problem maksymalnego przepływu można rozwiązać wielomianowo)

Dowód. Weźmy sobie $e = (u,v) \in E_f$. Spójrzmy na dwa kolejne momenty A, B, kiedy e stawała się nasycona. W momentach A,B $e \in E_f$. Ale po momencie A mamy $e \notin A$. Czyli kiedyś musiała się pojawić. Nazwijmy ten moment C.

Oczywiście $(v,u) \in E_f$ w momencie C i (v,u) leży na najkrótszej ścieżce $s \to t$. Zatem $\delta_{f'}(u) = \delta_{f'}(v) + 1$. Z poprzedniego lematu $\geqslant \delta_f(v) + 1 = \delta_f(u) + 1$

Pokazaliśmy, że pomiędzy A i B $\delta_f(u)$ rośnie o $\geqslant 2$, zatem tak się dzieje $\leqslant (V-1)/2$ razy. □

1.3 Algorytm Dinica i trzech hindusów

Znowu naturalny pomysł. Bezsensu odpalamy BFS-a wiele razy. Być może po updacie da się reużyć wynik naszego ostatniego BFS-a i updateować 'na jedno odpalenie BFS więcej ścieżek'

Definicja 8 (Przepływ blokujący)**.** *taki, że dla każdej ścieżki $s \to t$ istnieje krawędź nasycona.*

Będziemy robić mniej-więcej to samo, tylko w jednej iteracji będziemy szukać przepływu blokującego zamiast ścieżki powiększającej (blokującej).

Definicja 9 (Sieć warstwowa)**.** *L_f to jest pewien podgraf N_f taki, że:*

- $V(L_f) = V$

- $(u,v) \in E(L_f) \iff \delta_f(v) = \delta_f(u) + 1$

Schemat:

$f \leftarrow 0$
while $\exists$ ścieżka powiększająca **do**
  Zbuduj sieć warstwową L_f za pomocą BFS-a
  $b \leftarrow$ przepływ blokujący w L_f
  $f \leftarrow f + b$
  Zaktualizuj N_f
end while

Dlaczego to działa szybko? Będziemy ograniczać liczbę iteracji:

Lemat 7. *Niech b będzie przepływem blokującym w* L_f $\delta_{f+b}(t) > \delta_f(t)$

Dowód. Niech: $d = \delta_f(t)$
Pokażemy że dowolna ścieżka w $G_{f+b} : v_0, v_1, ...v_k$ ma długość $> d$.
Niech: $l(v_i) = \delta_f(v_i)$
Wtedy $\forall_{(v_i,v_{i+1})} \in G_{f+b} \rightarrow (v_i, v_{i+1}) \in G_f \rightarrow \delta_f(v_{i+1} \leqslant \delta_f(v) + 1)$ lub $(v_i, v_{i+1}) \notin G_f \rightarrow b$ przesłał po $(v_{i+1}, v) \rightarrow \delta_f(v_i) = \delta_f(v_{i+1}) + 1 \rightarrow l(v_{i+1}) = l(v_i) - 1$

Co najmniej raz zdarzył się przypadek drugi, bo b jest blokujące.

□

Zatem liczba iteracji $\leqslant V - 1$.

Pytanie pozostaje, jak szybko potrafimy znaleźć przepływ blokujący.

Będziemy robić dfs-a który jak znajdzie ścieżkę do t to skacze do s, jak się wycofuje to usuwa krawędź.

$b \leftarrow 0$
$v \leftarrow s$
$p \leftarrow \emptyset$
while $v! = sor\exists_w and(v, w \in E(L_f))$ **do**
  if $\exists_w (v,w) \in E(L_f)$ **then**
    ADVANCE:
    $push(p,(v,w))$
    $v \leftarrow w$
    if $v = t$ **then**
      AUGMENT:
      Powiększ b wzdłuż ścieżki powiększającej p
      $p \leftarrow \emptyset$
      $v \leftarrow s$
    else
      RETREAT:
      Usuń ostatnią krawędź $(u, w)zp$
      $E(L_f) \leftarrow E(L_f) \setminus (u,w)$
      $v \leftarrow u$
    end if
  end if
end while

Analiza: całkowity czas RETREAT = O(E) całkowity czas AUGMENT = O(VE) całkowity czas ADVANCE: każda operacja ADVANCE(v,w) w przyszłości owocuje RETREAT(v,w) lub AUGMENT, więc $\leqslant O(VE) + O(E)$

suma: $O(VE)$

Zatem: całkowity koszt algorytmu Dinica jest równy $O(V^2E)$

Trzech hindusów znajduje przepływ blokujący w czasie $O(V^2) \rightarrow$ cały przepływ w $O(V^3)$. Da się