Algorytm SPFA

Krzysztof PIECUCH *

„Dane są skierowany graf ważony $G = (V, E)$ z nieujemnymi wagami na krawędziach oraz dwa wyróżnione wierzchołki $s, f \in V$. Wagę krawędzi $u \to v \in E$ oznaczamy przez $w(u \to v)$. Należy znaleźć długość najkrótszej ścieżki z s do f."

O algorytmie Dijkstry można poczytać w książce: T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Wprowadzenie do algorytmów*.

Gdyby to oto zadanie dać do rozwiązania przeciętnemu polskiemu studentowi informatyki, zapewne zaimplementowałby algorytm Dijkstry. Gdyby jednak to samo zadanie zadać studentowi z Chin, to prawdopodobnie wykorzystałby do rozwiązania – w Polsce niemal nieznany – algorytm SPFA. Nic w tym dziwnego; jest on krótszy i dużo łatwiejszy do zaprogramowania.

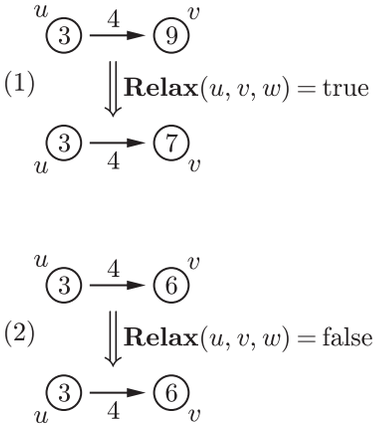
Często bywa tak, że uogólniając problem, łatwiej jest znaleźć jego rozwiązanie. Nie inaczej jest w tym przypadku. Zamiast szukać najkrótszej ścieżki z s do f, znajdziemy najkrótsze ścieżki z s do wszystkich wierzchołków $v \in V$.

Schemat większości algorytmów rozwiązujących problem najkrótszej ścieżki jest taki sam. Każdemu wierzchołkowi v przypisujemy wartość $d[v]$. Będzie ona oznaczać najmniejszą znalezioną dotychczas długość ścieżki z s do v. Początkowe wartości elementów tablicy d nadaje poniższa prosta procedura.

Initialize Single Source(G, s)
  for each $v \in V$ **do** $d[v] := \infty$;
  $d[s] := 0$;

Podstawową operacją, która modyfikuje wartość $d[v]$, jest relaksacja. Dla krawędzi $u \to v$ polega ona na sprawdzeniu, czy idąc tą krawędzią, dojdziemy do wierzchołka v szybciej, niż potrafiliśmy to zrobić do tej pory. Jeśli tak, to aktualizujemy wartość $d[v]$.

Relax(u, v)
  if $d[v] > d[u] + w(u \to v)$ **then**
    $d[v] := d[u] + w(u \to v)$;
    return true;
  else return false;



Relaksacja. W sytuacji (1) funkcja **Relax** znajduje krótszą ścieżkę do wierzchołka v i aktualizuje $d[v]$. W sytuacji (2) próba znalezienia krótszej ścieżki nie udaje się – żadna wartość nie zostaje zmieniona.

Algorytmy Dijkstry, Bellmana–Forda oraz SPFA różnią się tylko tym, w jakiej kolejności i ile razy relaksują krawędzie. SPFA (ang. *Shortest Path Faster Algorithm*) jest ulepszoną wersją algorytmu Bellmana–Forda. Zacznijmy więc od opisu tego algorytmu. Składa się on z $|V| - 1$ faz, a w każdej z nich relaksowane są wszystkie krawędzie grafu. Koszt czasowy algorytmu Bellmana–Forda to zatem $\Theta(VE)$.

Bellman–Ford(G, s)
  Initialize Single Source(G, s);
  for $i := 1$ **to** $|V| - 1$ **do**
    for each $u \to v \in E$ **do**
      Relax(u, v);

Skąd bierze się taka, a nie inna liczba faz algorytmu Bellmana–Forda? Podzielmy wszystkie wierzchołki grafu na grupy: $V_0 = \{s\}, V_1, V_2, \ldots$ W zbiorze V_i znajdują się wierzchołki, dla których pewna najkrótsza ścieżka z s składa się z i krawędzi i które nie znalazły się w żadnej grupie V_j dla $j < i$. Można pokazać przez łatwą indukcję, że po wykonaniu i-tej fazy algorytmu dla każdego z wierzchołków z V_i wartość w tablicy d jest ostateczna. Na koniec wystarczy zauważyć, że $V_i \neq \emptyset$ może zachodzić jedynie dla $i \leqslant |V| - 1$ (dlaczego?).

Zwróćmy teraz uwagę na następujące dwie sytuacje, w których relaksacja krawędzi $u \to v$ na pewno nie zmieni wartości $d[v]$:

a) wartość $d[u] = \infty$;
b) krawędź $u \to v$ była już relaksowana i od tego czasu wartość $d[u]$ nie uległa zmianie.

* student, Wydział Matematyki i Informatyki, Uniwersytet Wrocławski

Rozwiązanie zadania M 1231. Zauważmy, że dana w treści zadania równość jest równoważna zależności

$$(a+b+c+d)(a-c+2b-2d) = \\ = (a+c)(-a+b+c-d).$$

A zatem, gdyby liczba $a+b+c+d$ była liczbą pierwszą, to musiałaby być dzielnikiem jednej z liczb $a+c$ lub $-a+b+c-d$. Jednak nie jest to możliwe, gdyż $a+c < a+b+c+d$ oraz $|-a+b+c-d| < a+b+c+d$. Wobec tego liczba $a+b+c+d$ jest liczbą złożoną.

W algorytmie Bellmana–Forda często wykonuje się tego typu niepotrzebne relaksacje.

Algorytm SPFA korzysta ze struktury danych zwanej kolejką bez duplikatów. Można na niej wykonać następujące operacje:

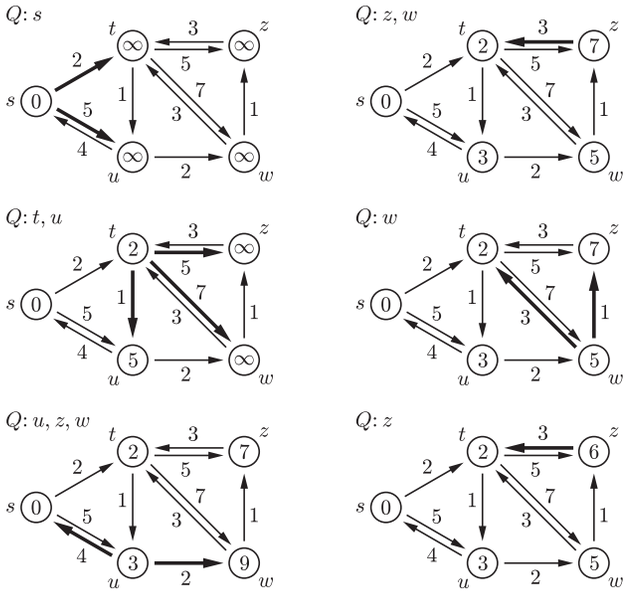
- **Init**(Q): tworzy pustą kolejkę bez duplikatów Q;
- **Enqueue**(Q, v): wstawia element v do kolejki Q, o ile Q jeszcze go nie zawiera;
- **Dequeue**(Q): wyjmuje z kolejki element, który był wstawiony najwcześniej;
- **IsEmpty**(Q): zwraca prawdę, jeśli kolejka jest pusta; w przeciwnym przypadku zwraca fałsz.

Jeśli przyjmiemy, że elementami Q są liczby z zakresu od 0 do $m-1$, to implementacja tej struktury danych może wyglądać następująco. Prócz zwykłej kolejki utrzymywać będziemy tablicę boolowską $T[0..(m-1)]$. Wartość $T[i]$ określa, czy element i znajduje się w kolejce, czy też nie. Trzeba teraz tylko, wstawiając i usuwając elementy kolejki, odpowiednio uaktualniać tablicę. W dalszej części artykułu kolejkę bez powtórzeń będziemy nazywać po prostu kolejką.

W algorytmie SPFA na kolejce Q przechowywane są wierzchołki grafu G. Wierzchołek u znajduje się w Q, jeśli spełnione są dwa warunki:

W tym miejscu polecamy uwadze Czytelnika parę podobnie wyglądających warunków a), b) z wcześniejszej części tekstu.

a) wartość $d[u] \neq \infty$;
b) krawędzie wychodzące z wierzchołka u nie były relaksowane lub od ostatniej relaksacji zmieniła się wartość $d[u]$.



Przykład działania algorytmu SPFA. Na rysunku pogrubiono krawędzie relaksowane. Choć $d[u]$ oraz $d[w]$ zostały zmienione dwukrotnie, to krawędzie wychodzące z tych wierzchołków relaksowaliśmy tylko raz. Było to spowodowane tym, że gdy zmieniliśmy po raz drugi wartość d tych wierzchołków, to były już one w kolejce. Natomiast wierzchołek z znalazł się w kolejce dwukrotnie, więc wychodząca z niego krawędź była relaksowana dwa razy.

Na początku jedynym wierzchołkiem spełniającym oba warunki jest s. Za każdym razem, gdy pobieramy z kolejki wierzchołek u, relaksujemy wszystkie krawędzie wychodzące z u. Jeśli relaksacja krawędzi $u \to v$ zmieni wartość $d[v]$, to wierzchołek v wstawiamy do kolejki. Algorytm kończy pracę, gdy kolejka staje się pusta.

```
SPFA(G, s)
  Initialize Single Source(G, s);
  Init(Q);
  Enqueue(Q, s);
  while not IsEmpty(Q) do
    u := Dequeue(Q);
    for each u → v ∈ E do
      if Relax(u, v) then
        Enqueue(Q, v);
```

Największą wadą tego algorytmu jest złożoność obliczeniowa. Dla pewnych grafów SPFA będzie działał tak samo kiepsko jak algorytm Bellmana–Forda, czyli w czasie $\Theta(VE)$ (czy potrafisz znaleźć przykład takiej rodziny grafów?). Jednak w praktyce algorytm radzi sobie bardzo dobrze. Wykonuje niewiele relaksacji, a dzięki prostej strukturze ma małą stałą. Do SPFA można też dodać pewne elementy losowości (na przykład przy wyborze kolejności relaksacji krawędzi wychodzących z ustalonego wierzchołka), co istotnie zmniejsza szansę na to, że dla danego grafu algorytm będzie bardzo nieefektywny.

Co jeszcze potrafi SPFA? Po dodaniu tablicy typu „z jakiego wierzchołka nastąpiła relaksacja” algorytm jest w stanie odtworzyć najkrótszą ścieżkę z s do dowolnego z wierzchołków. Dalej, na początku artykułu założyliśmy, że krawędzie grafu mają nieujemne wagi – bez tego warunku algorytm Dijkstry może nie zadziałać poprawnie. Jednak SPFA radzi sobie nie tylko z ujemnymi krawędziami, ale po drobnej modyfikacji wykryje również ujemne cykle w grafie. Jeśli natomiast chcemy obliczyć najkrótsze ścieżki między wszystkimi parami wierzchołków w grafie rzadkim, wystarczy uruchomić SPFA dla każdego możliwego wierzchołka startowego.

Dla grafu gęstego lepiej wykorzystać algorytm Floyda–Warshalla, także opisany we *Wprowadzeniu do algorytmów*.