Algorytmy i Struktury Danych

12: Znajdowanie najkrótszych ścieżek w grafach

©Marcin Sydow

Spis zagadnień

- Problem najkrótszych ścieżek z jednym źródłem
- Rozwiązanie "sznurkowe"
- Warianty
- Relaksacja krawędzi
- Wariant 1: DAG
- Wariant 2: nieujemne krawędzie (Dijkstra)
- Wariant 3: dowolny graf (Bellman-Ford)

Problem najkrótszych ścieżek

Wejście: skierowany graf $G = (V, E)$ z wagami na krawędziach, danymi przez funkcje $w : E \rightarrow R$, i wierzchołek startowy $s \in V$

Wyjście: dla każdego wierzchołka $v \in V$

- długość najkrótszej ścieżki $\mu(s, v)$ z s do v, jeśli istnieje
- rodzic w drzewie najkrótszych ścieżek (jeśli istnieje), pozwalający zrekonstruować najkrótsze ścieżki z s

Najkrótsza ścieżka może nie istnieć z dwóch powodów:

- v może nie być osiągalny z s
- w grafie mogą istnieć *ujemne cykle* (wtedy może nie być dolnego ograniczenia na długość ścieżki)

(przykład)

Warianty

Przy projektowaniu optymalnego algorytmu, można wziąć pod uwagę pewne specjalne własności grafu, np.:

- graf jest skierowany, albo nieskierowany
- graf jest acykliczny (wtedy można zastosować najszybszy algorytm)
- wagi są nieujemne (wtedy można zastosować szybszy algorytm)[1]

Pokazane zostaną różne warianty w zależności od własności wejściowego grafu

[1] albo całkowite - w tym przypadku jest bardziej efektywna struktura danych

Najkrótsze ścieżki - własności

Załóżmy dla grafu $G = (V, E)$ i wierzchołków $s, v \in V$ następującą konwencję: $\mu(s, v)$ oznacza długość najkrótszej ścieżki z s do v w G, czasami, jeśli s jest znane z kontekstu, będziemy pisali krócej $\mu(v)$:

- $\mu(s, v) = +\infty$ (kiedy nie ma ścieżki z s do v)
- $\mu(s, v) = -\infty$ (kiedy istnieje ścieżka z s do v zawierająca ujemny cykl)
- $\mu(s, v) = d \in R$ (w pozostałych przypadkach)

Lemat:
Podścieżka najkrótszej ścieżki musi być najkrótszą ścieżką

Obliczanie najkrótszych ścieżek: idea

Ogólny pomysł jest podobny do BFS (ze źródła s). Z każdym wierzchołkiem v związujemy 2 atrybuty:

- `v.distance`: przechowuje najkrótszą (obecnie znaną) odległość z s do v
- `v.parent`: przechowuje poprzednika (rodzica) v na najkrótszej ścieżce (obecnie znanej) z s

inicjalizacja: `s.distance = 0`, `s.parent = s`, a wszystkie pozostałe wierzchołki mają atrybut `distance` ustawiony na $+\infty$ oraz `parent` na `null`.

Uaktualnienia tych atrybutów są następnie "propagowane" przez krawędzie: nazywane jest to **relaksacją** krawędzi

Relaksacja krawędzi


```
%% relax((u,v))              # (u,v) jest krawędzią w grafie
  if u.distance + w(u,v) < v.distance
    v.distance = u.distance + w(u,v)
    v.parent = u
```

Algorytmy dokonują kolejnych relaksacji dopóki najkrótsze ścieżki nie zostaną obliczone lub ujemny cykl wykryty.

Relaksacje mają pewne ważne własności, np:
(założywszy uprzednie dokonanie opisanej inicjalizacji)
- po dowolnym ciągu relaksacji $\forall v \in V$ `v.distance` $\geq \mu(v)$
czyli, że wartość `distance` odkryta przez relaksacje nie może spaść poniżej prawdziwej wartości odległości (dowód przez indukcję po liczbie relaksacji krawędzi)

Poprawność relaksacji

Lemma

Po dokonaniu ciągu R relaksacji takiego, że zawiera on (jako podciąg) najkrótszą ścieżkę $p = (e_1, ..., e_k)$ z s do v, zachodzi, że: $v.distance = \mu(s, v)$ (czyli, że najkrótsza ścieżka zostanie odkryta).

Dowód: Ponieważ p jest najkrótszą ścieżka, więc mamy $\mu(v) = \sum_{1 \leq j \leq k} w(e_j)$. Niech v_i oznacza koniec krawędzi e_i, dla $0 < i \leq k$ ($v_0 = s$). Pokażemy przez indukcję, że po i-tej relaksacji zachodzi $v_i.distance \leq \sum_{1 \leq j \leq i} w(e_j)$. Jest to prawda na początku (s.distance == 0). Następnie, po i-tej relaksacji, $v_i.distance \leq v_{i-1}.distance + w(e_i) \leq \sum_{1 \leq j \leq i} w(e_j)$ (z defincji relaksacji i na mocy indukcji). Więc po k-tej relaksacji zachodzi: $v.distance \leq \mu(v)$. Ale $v.distance$ nie może być niższe nić $\mu(v)$ (poprzedni lemat), a więc zachodzi $v.distance == \mu(v)$.

Sortowanie topologiczne (tylko grafy skierowane)

Sortowanie topologiczne grafu skierowanego polega na takim ustawieniu wierzchołków grafu w ciąg, że jeśli w grafie jest krawędź (u, v), to u musi być przed v (ustawienie "zgodne" z krawędziami).

Graf da się posortować topologicznie $\Leftrightarrow$ nie zawiera cykli (skierowanych).

Można efektywnie posortować topologicznie graf w następujący sposób:

- zastosować DFS
- ustawić wierzchołki od największego czasu zakończenia do najmniejszego

Uwaga: istnieje też prostsze koncepcyjnie rozwiązanie przez iteracyjne usuwanie wierzchołków o stopniu wejściowym 0, też można zaimplementować w czasie liniowym

1: Najkrótsze ścieżki w grafie acyklicznym (DAG)

Jeśli graf jest skierowanym grafem acyklicznym (DAG), to stanowi to bardzo prosty przypadek dla problemu najkrótszych ścieżek ze źródła s.

Każdy DAG może być topologicznie posortowany (np. przez DFS) w czasie $O(m+n)$, co daje ciąg wierzchołków $(v_1, ..., v_n)$. Następnie, zakładając, że $s = v_j$, dla pewnego $0 < j \leq n$, można dokonać relaksacji wszystkich krawędzi wychodzących z v_j, a następnie wychodzących z v_{j+1} i tak dalej aż do v_n.

W ten sposób każda krawędź poddana jest relaksacji co najwyżej raz a każda najkrótsza ścieżka jest odkryta jako podciąg ciągu dokonanych relaksacji. Ponieważ złożoność czasowa relaksacji jest stała, więc algorytm ma łączną złożoność $O(m+n)$ w tym przypadku.
UWAGA: wierzchołki występujące przed s w posortowanym ciągu są nieosiągalne z s.

2: Algorytm Dijkstry (wagi nieujemne)

Zauważmy, że jeśli nie ma krawędzi o ujemnych wagach, to nie ma też ujemnych cykli. Natomiast cykle mogą istnieć, więc nie można założyć wykonalności sortowania topologicznego.

Pomysł algorytmu w tym przypadku polega na dokonywaniu relaksacji w kolejności niemalejących najkrótszych odległości od źródła. Dzięki nieujemności wag, każda najkrótsza ścieżka zostanie w ten sposób odkryta.
Uwaga: żeby osiągnąć powyższą kolejność, w algorytmie stosowana jest *kolejka priorytetowa* przechowująca kolejne wierzchołki do odwiedzenia (priorytetem jest wartość atrybutu `distance`)

Przykład (sznurki z węzełkami):
Jest to analogiczne do podnoszenia ze stołu sznurków powiązanych za pomocą węzełków (kolejne podnoszone ze stołu węzełki odpowiadają wierzchołkom).

Algorytm (Dijkstra)

(pq - kolejka priorytetowa z operacją decreaseKey, priorytetem jest wartość atrybutu distance)

```
s.distance = 0
pq.insert(s)
s.parent = s

for-each v in V except s:
   v.distance = INFINITY
   v.parent = null

while(!pq.isEmpty())
   scannedNode = pq.delMin()
   for-each v in scannedNode.adjList:
      if (v.distance > scannedNode.distance + w(scannedNode, v))
         v.distance = scannedNode.distance + w(scannedNode, v)
         v.parent = scannedNode
         if (pq.contains(v)) pq.decreaseKey(v)
         else pq.insert(v)
```

(żeby efektywnie zaimplementować operacje contains i decreaseKey musimy użyć tzw. *adresowalnej* kolejki priortytowej, czyli wyposażonej w dodatkowy słownik mapujący wierzchołki do ich pozycji w kolejce priorytetowej)

Analiza pesymistczyna algortymu Dijkstry

rozmiar danych: $n = |V|$, $m = |E|$

operacja dominująca: porównanie priorytetów (również wewnątrz kolejki priorytetowej), aktualizacja atrybutów

Złożoność głównej pętli zależy od użytej implementacji kolejki priorytetowej.
inicjalizacja: $O(n)$

pętla: $O(n \times (\text{delMin} + \text{insert}) + m \times \text{decreaseKey})$

$O(nlogn) + O(mlogn) = \mathbf{O((n + m)logn)}$ (dla kopca binarnego)

Analiza c.d.

Powyższa analiza dotyczy przypadku pesymistycznego. Można pokazać, że w przypadku *przeciętnym* liczba operacji `decreaseKey` wynosi $O(nlog(m/n))$, co daje *przeciętną* złożoność czasową $O(m + nlog(m/n)logn)$.
Przy dostatecznie gęstym grafie (gdy pierwszy czynnik dominuje nad drugim) otrzymujemy *liniowy* czas przeciętny.
Ponadto, można użyć kopca Fibonacciego, który ma amortyzowany koszt *stały* operacji `decreaseKey` i wtedy otrzymujemy czas pesymistyczny (jednak kopiec Fibonacciego ma ukrytą wyższą stałą multiplikatywną):
$O(m + nlogn)$

Ponadto, jeśli wagi są całkowite i ograniczone przez stałą C, można zredukować złożoność pesymistyczną do:

$$O(m + nC)$$

stosując tzw. *monotoniczną bukietową* kolejkę priorytetową.

3: Algorytm Bellmana-Forda (dowolne wagi)

W poprzednich przypadkach wystarczało conajwyżej m relaksacji.

Zawsze jednak wystarcza dokonać $O(nm)$ relaksacji, aby odkryć *każdą* najkrótszą ścieżkę (jako podciąg). Jest to rodzaj podejścia "siłowego", które działa dla każdego grafu.

Ponieważ dowolna najkrótsza ścieżka może zawierać conajwyżej $n-1$ krawędzi spośród m, więc wystarczy dokonać $(n-1)$-krotnej relaksacji wszystkich m krawędzi ustawionych w ustalony ciąg, aby ciąg krawędzi każdej najkrótszej ścieżki był w nim zawarty jako podciąg (i w ten sposób wykryć wszystkie dobrze określone najkrótsze ścieżki). Dla wierzchołków nieosiągalnych będzie $v.d == \infty$. Aby następnie wykryć jakiekolwiek ujemne cykle, wystarczy następnie jeszcze raz dokonać m relaksacji (te, dla których atrybut d wciąż maleje, leżą na ścieżkach zawierających ujemny cykl) i następnie w czasie liniowym przypisać wierzchołkom osiągalnym z tego cyklu wartość $v.d == \infty$

Bellman-Ford's Algorithm


```
%% (initialise as in Dijkstra)

for(i = 1; i <= (n-1); i++)
   for each e in E
      relax(e)

for each e=(u,v) in E
   if (u.distance + w(u,v) < v.distance)
      identifyNegativeCycle(v)

***

identifyNegativeCycle(v)
   if (v.distance > -infinity)
      v.distance = -infinity
      for each w in v.adjList
         identifyNegativeCycle(w)
```

Podsumowanie

- Problem najkrótszych ścieżek z jednym źródłem
- Rozwiązanie "sznurkowe"
- Warianty
- Relaksacja krawędzi
- Wariant 1: DAG
- Wariant 2: nieujemne krawędzie (Dijkstra)
- Wariant 3: dowolny graf (Bellman-Ford)
- Najkrótsze ścieżki dla wszystkich par

Przykładowe Zadania

- podaj specyfikację problemu najkrótszych ścieżek z jednym źródłem
- podaj przynajmniej 2 przykłady zastosowań problemu najkrótszych ścieżek
- na czym polega relaksacja krawędzi
- podaj specyfikację problemu sortowania topologicznego
- kiedy można posortować topologicznie graf? Jak można to zrobić (podaj 2 sposoby)
- mając dany DAG dokonaj jego posortowania topologicznego (używając DFS)
- mając dany graf uzasadnij, który z 3 omawianych algorytmów będzie najbardziej efektywny dla tego grafu
- mając dany graf z nieujemnymi wagami zastosuj algorytm Dijkstry i przypisz wszystkim atrybutom wierzchołków wartości zgodnie z algorytmem
- zastosuj algorytm Bellmana-Forda do podanego grafu
- dokonaj analizy złożoności czasowej 3 omówionych algorytmów

Dziękuję za uwagę