Grafy i Zastosowania

6: Najkrótsze ścieżki

©Marcin Sydow

Spis zagadnień

Najkrótsze Ścieżki

Warianty

Relaksacja

DAG

Algorytm Dijkstry

Bellman-Ford

Wszystkie pary

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
  - Zmiana wag grafu na nieujemne przez potencjały
  - Algorytm Johnsona
  - Przykład: oszacowanie średnicy grafu

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 $v.distance == \mu(v)$.

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)
```

Najkrótsze ścieżki dla wszystkich par

Problem: wejście: dany jest graf skierowany z wagami na krawędziach.
wyjście: najkrótsze ścieżki pomiędzy wszystkimi parami wierzchołków

Rozwiązanie 1: wykonać Bellmana-Forda ze wszystkich wierzchołków ($O(n^2 m)$)

Rozwiązanie 2 (szybsze): odpowiednio zamienić wagi na nieujemne (zachowując najkrótsze ścieżki) i wykonać algorytm Dijkstry ze wszystkich wierzchołków ($O(n(m + nlogn))$)

Uwaga: rozwiązanie 2 da się wykonać tylko jeśli nie ma cykli ujemnych.

Zmiana wag na nieujemne

Aby móc zastosowań algorytm Dijkstry, w grafie zostają zmienione wagi na nieujemne, tak aby zachować najkrótsze ścieżki. Mianowicie, każdej krawędzi $e = (v, w)$ o dotychczasowej wadze $w(e)$ przypisywana jest nowa waga:

$$w'(e) = w(e) + pot(v) - pot(w),$$

gdzie $pot()$ jest pewną funkcją nieujemną mającą pewne pożądane własności:

- wszystkie nowe wagi są nieujemne
- najkrótsze ścieżki przy nowych wagach są takie same jak przy starych
- sumy wag na cyklach nie ulegają zmianie

Funkcję spełniającą te warunki nazywamy *potencjałem* wierzchołka.

Potencjał wierzchołka

Aby funkcja $pot()$ spełniała powyższe warunki wyznaczamy ją następująco:

- dodajemy do grafu pewien sztuczny wierzchołek s oraz łączymy go krawędziami (skierowanymi) z wagami 0 do wszystkich wierzchołków grafu
- obliczamy najkrótsze odległości $\mu(v)$ z s do każdego wierzchołka v grafu

przykład

Nowe wagi zachowują najkrótsze ścieżki

Lemat 1:

Jeśli p, q są ścieżkami z v do w, a $c(p)$ oznacza koszt tej ścieżki przy oryginalnych wagach, a $c'(p)$ przy nowych wagach, to:

1. $c'(p) = pot(v) + c(p) - pot(w)$
2. $c'(p) \leq c'(q)$ wtw $c(p) \leq c(q)$ (zachowuje porządek kosztów ścieżek)
3. najkrótsze ścieżki przy nowych kosztach są takie same jak przy starych

Dowód: Niech $p = (e_1, \ldots, e_{k-1})$, $e_i = (v_i, v_{i+1})$, $v = v_0$ i $w = v_k$. Wtedy:
$c'(p) = \sum_{i=0}^{k-1} w'(e_i) = \sum_{0 \leq i < k} (pot(v_i) + w(e_i) - pot(v_{i+1})) =$
$pot(v_0) + \sum_{0 \leq i < k} w(e_i) - pot(v_k) = pot(v_0) + c(p) - pot(v_k)$

(a więc zmiana kosztu ścieżki zależy tylko od wierzchołków końcowych) Wynika z tego punkt 2 i 3 Lematu.

Wniosek:

Dla każdego cyklu C, $c'(C) = c(C)$ (nie zmienia kosztów cykli)

Nowe wagi są nieujemne

Lemat 2:
Jeśli graf nie ma ujemnych cykli i wszystkie wierzchołki są osiągalne z s, to przy zdefiniowaniu potencjału $pot(v) = \mu(v)$ (jako długości najkrótszej ścieżki z s) nowe wagi są nieujemne dla wszystkich krawędzi.

Dowód:
Ponieważ nie ma ujemnych cykli, to wszystkie wartości $pot(v)$ są dobrze zdefiniowane. Rozważmy dowolną krawędź $e = (v, w)$. Mamy tedy: $\mu(v) + w(e) \geq \mu(w)$ (gdyż najkrótsze ścieżki spełniają nierówność trójkąta) a więc: $w'(e) = \mu(v) + w(e) - \mu(w) \geq 0$

Uwaga: czy wag nie można po prostu zwiększyć o stałą wartość, tak aby były nieujemne? (podaj odpowiedni kontrprzykład)

Pomysł algorytmu Johnsona

Z powyższych dwóch lematów wynika, że nowe wagi zachowują najkrótsze ścieżki i pozwalają zastosować algorytm Dijkstry.

Aby obliczyć nowe wagi, należy najpierw wykonać algorytm Bellmana-Forda z wierzchołka s aby obliczyć wartość $\mu(v)$ dla każdego wierzchołka v (przy okazji pozwala on wykryć ewentualne ujemne cykle).

Następnie wykonać algorytm Dijkstry z każdego wierzchołka i przywrócić poprzednie wagi.

Taki algorytm ma złożoność: $O(mn)$ (BF) + $O(n(m + nlogn))$ (n razy Dijkstra, przy użyciu kopca Fibonacciego), czyli łącznie: $O(nm + n^2 logn)$

Algorytm Johnsona


```
%% algorytmJohnsona(){
dodaj wierzchołek s z krawędziami o zerowej wadze do wszystkich wierzcho

wykonaj Bellman-Ford aby obliczyć najkrótsze ścieżki z s

for each node v:
   przypisz pot(v) jako odległość od s

for each node v:
   wykonaj Dijkstra z v na zmienionych wagach
   (bo są nieujemne)

for each edge e=(u,v):
   przywróć poprzednie wagi (**)
   (ponieważ najkrótsze ścieżki są te same)
}
```

$(**)\ w(v,w) = w'(v,w) + pot(w) - pot(v)$

Przykład: oszacowanie średnicy grafu ($diam(G)$)

(przypomnienie: maksymalna odległość pomiędzy parą wierzchołków w grafie)

Można prosto oszacować średnicę grafu nie licząc wszystkich par odległości na podstawie poniższej obserwacji:

Lemat:
Niech $D'(s) = max_{u \in V} \mu(s, u)$, dla pewnego wierzchołka s (ekscentryczność wierzchołka s).

- Jeśli graf jest nieskierowany, to zachodzi $D'(s) \leq diam(G) \leq 2D'(s)$.
- Jeśli graf jest skierowany, to powyższa nierówność nie zachodzi, ale zachodzi $max(D'(s), D''(s)) \leq D \leq D'(s) + D''(s)$, gdzie $D''(s) = max_{u \in V} \mu(u, s)$.

Czyli średnicę można oszacować przez ekscentryczność dowolnego wierzchołka (czyli złożoność liniowa zamiast kwadratowej)

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
  - Zmiana wag grafu na nieujemne przez potencjały
  - Algorytm Johnsona
  - Przykład: oszacowanie średnicy grafu

Przykładowe Zadania

- wykonaj efektywny algorytm obliczania najkrótszych ścieżek z podanego źródła dla podanego grafu (najpierw zaklasyfikuj graf do odpowiedniego przypadku)
- dokonaj zmiany wag podanego grafu na nieujemne zachowując najkrótsze ścieżki, zgodnie z przedstawioną techniką potencjałów
- wykonaj algorytm Johnsona dla podanego grafu
- oszacuj średnicę podanego grafu (nieskierowanego lub skierowanego) za pomocą przedstawionej techniki.

Dziękuję za uwagę