

Problem najkrótszych ścieżek – wprowadzenie

Warianty problemu SHORTEST PATH

- 1 $G = (V, E)$ jest DAGiem
- 2 G jest dowolny, $c(u, v) = 1$ dla wszystkich $(u, v) \in E$
- 3 dane są $s \in V$ oraz $c(u, v) \geq 0 \rightarrow$ znajdź najkrótsze ścieżki od s do wszystkich $v \in V$ (SINGLE-SOURCE SHORTEST PATH)
- 4 jak w 3, ale $c(u, v)$ są dowolne (możliwe ujemne długości)
- 5 znajdź najkrótsze ścieżki między każdą parą wierzchołków (ALL-PAIRS SHORTEST PATH)
- 6 różne uogólnienia
 - np. dodatkowe parametry – najkrótsza ścieżka, dla której średni czas przejazdu nie przekracza podanego limitu, itp.

- Problemy najkrótszych ścieżek mają szereg różnych zastosowań – patrz np. rozdział 4.2 w [AMO14].
- Na kursie AiSD omawiamy problemy 1–4 (podstawy).
- Bardziej szczegółowo temat ten omawiany jest na kursie wybieralnym *Algorytmy optymalizacji dyskretnej*.

Notatki

Krawędzie o długości 1 – algorytm BFS

Algorytm BFS (breadth-first search)

- przeglądanie grafu „warstwa po warstwie”
 - zaczynamy od s ($dist = 0$)
 - następnie przeglądamy po kolei jego sąsiadów ($dist = 1$)
 - następnie przeglądamy jeszcze nieodwiedzonych sąsiadów tych sąsiadów ($dist = 2$) itd.
- w iteracji d przeglądamy węzły odległe o d od węzła s
- w iteracji $d + 1$ odwiedzamy węzły odległe o $d + 1$ od s , tj. sąsiadów węzłów przeglądanych w iteracji d , którzy nie zostali wcześniej odwiedzeni
- wykorzystujemy **kolejkę FIFO Q**
 - początkowo w Q jest wierzchołek s
 - w każdym kroku ściągamy wierzchołek v z początku Q i dodajemy na jej koniec wszystkich **jeszcze nieodwiedzonych** sąsiadów v
 - kończymy, gdy kolejka Q jest pusta
- dla każdego węzła v pamiętamy jego odległość $dist(v)$ od s oraz poprzednika $pred(v)$ w drzewie BFS
 - początkowo $dist(s) = 0$ oraz $dist(v) = \infty$ dla $v \neq s$

Notatki

Algorytm BFS

- **Input:** graf $G = (V, E)$, $s \in V$ – wierzchołek początkowy
- **Output:**
 - $dist(v)$, $v \in V$ – długości najkrótszych ścieżek od s ; $dist(v) = \infty$ gdy v nie jest osiągalny z s
 - $pred(v)$, $v \in V$ – poprzednicy w drzewie BFS

procedure BFS($G = (V, E)$, s)

```

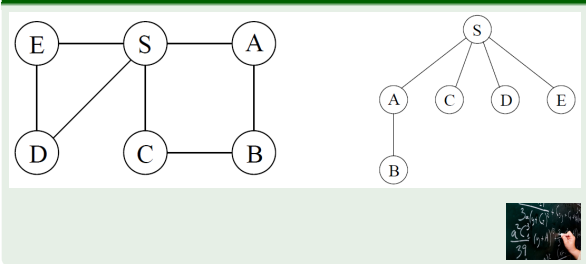
1: for all  $v \in V$  do  $dist(v) \leftarrow \infty$   $pred(v) \leftarrow null$ 
2:  $dist(s) \leftarrow 0$ 
3:  $Q \leftarrow [s]$ 
4: while  $Q$  is not empty do
5:    $u \leftarrow pull(Q)$  ▷ wybór przeglądanego wierzchołka
6:   for all  $(u, v) \in E$  do
7:     if  $dist(v) = \infty$  then ▷  $v$  nie był wcześniej odwiedzony
8:        $push(Q, v)$  ▷ dodaj  $v$  na koniec kolejki
9:        $dist(v) \leftarrow dist(u) + 1$  ▷ ustaw odległość  $v$  od  $s$ 
10:       $pred(v) \leftarrow u$  ▷ ustaw poprzednika  $v$  w drzewie BFS

```

Notatki

Przykład

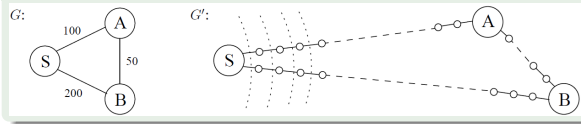
Przykład [DPV06]



Notatki

Ulepszenie algorytmu naiwnego

Przykład [DPV06]



Jak ulepszyć algorytm „naiwny”? – Intuicje

- dopóki BFS w G' nie dotrze do któregoś z wierzchołków z G , „nic ciekawego się nie dzieje”
- w każdej iteracji BFS w G' dla każdego wierzchołka v w G mamy pewne ograniczenie górne $T(v)$ na „czas dotarcia” do v
 - początkowo $T(s) = 0$, $T(v) = \infty$ dla $v \neq s$
 - w kolejnym kroku mamy $T(v) = c(s, v)$ dla wszystkich sąsiadów s (ich potencjalnym poprzednikiem jest s)
 - są to ograniczenia górne, bo w trakcie przechodzenia grafu możemy znajdować „skrótty”
- ...

Notatki

Algorytm Dijkstry – wprowadzenie

Jak ulepszyć algorytm „naiwny”? – Intuicje

- [...] pierwsza „ważna rzecz” dzieje się po dotarciu do $u \neq s$ o **najmniejszym** estymowanym czasie dotarcia $T(u)$
- ponieważ długości krawędzi $c(\cdot, \cdot) \geq 0$, do u nie możemy dotrzeć wcześniej $\rightarrow dist(u) = T(u)$, $pred(u) = s$ i najkrótsza ścieżka od s do u jest już wyznaczona
- z u mogą wychodzić krawędzie do pewnych v t. że:
 - $T(v) = \infty \rightarrow$ dla v możemy ustawić estymowany czas dotarcia na $T(v) = dist(u) + c(u, v)$ oraz potencjalnego poprzednika na u
 - $T(v) < \infty \rightarrow$ sprawdzamy, czy nie znaleźliśmy krótszej ścieżki od s do v . Jeśli $T(v) > dist(u) + c(u, v)$, to szybciej dotrzemy do v idąc przez u , więc uaktualniamy (zmniejszamy) estymowany czas dojścia do v i jego potencjalnego poprzednika
- kolejna „ważna rzecz” dzieje się w momencie, gdy dotrzemy do wierzchołka u' o **najmniejszym** estymowanym czasie dotarcia $T(u')$, do którego najkrótsza ścieżka nie jest jeszcze ustalona
 - powtarzamy procedurę, aż wyznaczymy najkrótsze ścieżki do wszystkich wierzchołków osiągalnych z s

Notatki

Algorytm Dijkstry – wprowadzenie

Idea algorytmu Dijkstry

- algorytm iteracyjnie oblicza $dist(v)$ dla $v \in V$
- w każdym kroku dzieli (niejawnie) V na zbiory S oraz $\bar{S} = V \setminus S$
 - S – wierzchołki z wyznaczonymi najkrótszymi ścieżkami od s (permanentne, optymalne wartości $dist(\cdot)$)
 - \bar{S} – wierzchołki z pewnym ograniczeniem górnym na długość najkrótszej ścieżki od s (tymczasowe wartości $dist(\cdot)$)
- permanentne wartości $dist(v)$ ustalane są w kolejności według rosnącej odległości węzłów od s

Algorytm Dijkstry w każdym kroku wybiera $u \in \bar{S}$ z najmniejszą tymczasową wartością $dist(u)$ (zaczynając od s), oznacza $dist(u)$ jako optymalne („przenosi u z \bar{S} do S ”) i wykonuje **distance update**.

Kluczowa obserwacja

Dla $u \in \bar{S}$ o najmniejszej tymczasowej wartości $dist(u)$, ta wartość jest optymalna i może zostać uznana za permanentną (docelową).

Notatki

Algorytm Dijkstry – implementacja generyczna

- **Input:**
 - graf $G = (V, E)$ z wagami $c(u, v) \geq 0$ dla $(u, v) \in E$
 - $s \in V$ – wierzchołek początkowy
- **Output:**
 - $dist(v)$, $v \in V$ – długości najkrótszych ścieżek od s
 - $pred(v)$, $v \in V$ – poprzednicy w drzewie najkrótszych ścieżek

```

1:  $S \leftarrow \emptyset$ ,  $\bar{S} \leftarrow V$ 
2: for all  $v \in V$  do  $dist(v) \leftarrow \infty$ ,  $pred(v) \leftarrow null$ 
3:  $dist(s) \leftarrow 0$ 
4: while  $\bar{S} \neq \emptyset$  do
5:   select  $u \in \bar{S}$  with smallest  $dist(\cdot)$  ▷ node selection
6:    $S \leftarrow S \cup \{u\}$ ,  $\bar{S} \leftarrow \bar{S} \setminus \{u\}$ 
7:   for all  $(u, v) \in E$  do
8:     if  $dist(v) > dist(u) + c(u, v)$  then ▷ distance update
9:        $dist(v) \leftarrow dist(u) + c(u, v)$ 
10:       $pred(v) \leftarrow u$ 

```

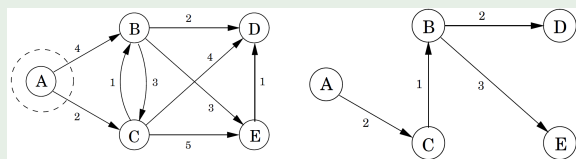
Notatki

Algorytm Dijkstry – Przykład

Przykład

Pojedyncza iteracja (dopóki $\bar{S} \neq \emptyset$)

- wybierz $u \in \bar{S}$ z najmniejszą tymczasową wartością $dist(u)$
- przenieś u z \bar{S} do S
- wykonaj operację *distance update*:
 $(\forall (u, v) \in E)(dist(v) \leftarrow \min\{dist(v), dist(u) + c(u, v)\})$



Notatki

Algorytm Dijkstry – obserwacje

Najważniejsze obserwacje

- algorytm Dijkstry zatrzymuje się, gdy wyznaczy najkrótsze ścieżki do wszystkich wierzchołków osiągalnych z s (wszystkie wartości $dist(v)$ są optymalne)
- w kolejnych iteracjach wyznaczane jest (niejawnie) drzewo o korzeniu w s złożone z wierzchołków $v \in V$, dla których $dist(v) \neq \infty$ (permanentne i tymczasowe)
- na koniec będzie to drzewo najkrótszych ścieżek
- to drzewo możemy odtworzyć na podstawie wskaźników $pred(v)$

Kluczowe operacje

- node selection** – wybór wierzchołka $u \in \bar{S}$ t. że $dist(u) = \min\{dist(v) : v \in \bar{S}\}$
- distance update** – dla wybranego wierzchołka u przeglądamy wszystkie krawędzie $(u, v) \in E$ i wyznaczamy $dist(v) = \min\{dist(v), dist(u) + c(u, v)\}$

Notatki

Generyczny algorytm Dijkstry – złożoność

Czas działania ($|V| = n, |E| = m$)

Główne operacje: *node selection* oraz *distance update*

- node selection**: $O(n^2)$
 - łącznie n operacji
 - czas pojedynczego wyboru ograniczony z góry przez $|\bar{S}| \leq n$
- distance update**: $O(m)$
 - dla każdego $u \in V$ robimy *update* dla każdej krawędzi (u, v)
 - jedna operacja o koszcie $O(1)$ dla m krawędzi

Razem: $O(n^2 + m)$

Uwagi i optymalizacje

- dla grafów gęstych $m = \Theta(n^2)$ asymptotycznie lepiej się nie da
 - musimy przejrzeć każdą krawędź \rightarrow składnik $O(m)$ zostanie
- da się poprawić złożoność dla grafów rzadkich z $m = o(n^2)$
- miejsce do optymalizacji: operacja *node selection*
 - albo poprawiamy worst-case complexity do $o(n^2)$
 - albo znacząco redukujemy czas dla „praktycznych instancji”

Notatki

Algorytm Dijkstry – implementacje kopcowe

Optymalizacja operacji node selection

- w każdej iteracji wybieramy wierzchołek u z najmniejszą tymczasową wartością $dist(u)$
- pomysł**: użyj **kolejki priorytetowej** (typu min)
 - kluczem jest tymczasowa wartość $dist(\cdot)$
 - w kolejce przechowujemy jedynie wierzchołki z \bar{S}
 - nie musimy przechowywać wierzchołków $v \in S$ z $dist(v) = \infty$
- efektywne implementacje kolejek priorytetowych – **kopce**

Kolejka priorytetowa

- każdy element posiada wartość liczbowa – klucz (priorytet)
- używamy kolejki typu **min**
- wspierane operacje („minimalny interfejs” kolejki priorytetowej):
 - `create-queue()`
 - `find-min(Q)`
 - `extract-min(Q)`
 - `insert(Q, i, k)`
 - `decrease-key(Q, i, k)`

Notatki

Algorytm Dijkstry – implementacje kopcowe

procedure Heap-Dijkstra($G = (V, E), c, s$)

```
1:  $H \leftarrow \text{create-heap}()$  ▷ klucze – wartości  $dist(\cdot)$ 
2: for all  $v \in V$  do  $dist(v) \leftarrow \infty, pred(v) \leftarrow \text{null}$ 
3:  $dist(s) \leftarrow 0$ 
4: insert( $H, s, dist(s)$ )
5: while  $H$  is not empty do
6:    $u \leftarrow \text{extract-min}(H)$  ▷ node selection
7:   for all  $(u, v) \in E$  do
8:     if  $dist(v) > dist(u) + c(u, v)$  then ▷ distance update
9:        $prev\_dv \leftarrow dist(v)$ 
10:       $dist(v) \leftarrow dist(u) + c(u, v)$ 
11:       $pred(v) \leftarrow u$ 
12:      if  $prev\_dv = \infty$  then
13:        insert( $H, v, dist(v)$ )
14:      else
15:        decrease-key( $H, v, dist(v)$ )
```

AlSD – Wykład 20. Ścieżki w grafach

Notatki

Złożoność algorytmu Dijkstry

Implementacja kopcowa – złożoność ($|V| = n, |E| = m$)

- każdy wierzchołek osiągalny z s wkładamy do kopca i wyciągamy z niego dokładnie jeden raz
- mamy więc po $\leq n$ operacji extract-min oraz insert
- każdą krawędź ze spójnej składowej s przeglądamy jeden raz – $O(m)$ operacji decrease-key
- **złożoność zależy od użytej implementacji kopca**

Kopiec binarny ($|V| = n, |E| = m$)

- insert, extract-min i decrease-key: $O(\log n)$
- **BinaryHeap-Dijkstra**: $O((m+n)\log n)$
 - $O(m\log n)$ gdy graf jest spójny
 - dla $m = \Theta(n^2)$ mamy $O(n^2 \log n)$ – gorzej niż wersja generyczna (zwykła tablica zamiast kopca)
 - dla $m = \Theta(n)$ mamy $O(n\log n)$ – dużo lepiej

AlSD – Wykład 20. Ścieżki w grafach

Notatki

Złożoność algorytmu Dijkstry

Kopiec d -arny

- insert i decrease-key: $O(\log_d n)$, extract-min: $O(d \log_d n)$
- **d -aryHeap-Dijkstra**: $O(m \log_d n + n d \log_d n)$
- Jakie d jest asymptotycznie optymalne?
 - $m \log_d n = n d \log_d n \iff d = \max\{2, \lceil m/n \rceil\}$
 - dostajemy wówczas złożoność $O(m \log_d n)$
 - gdy $m = \Omega(n^{1+\epsilon})$ dla pewnego $\epsilon > 0$, to $d = \lceil m/n \rceil = \Omega(n^\epsilon)$ i mamy złożoność $O(m)$ (algorytm asymptotycznie optymalny)

Kopiec Fibonacciego

- extract-min: $O(\log n)$
- pozostałe: $O(1)$ (**koszt zamortyzowany**)
- **FibonacciHeap-Dijkstra**: $O(m + n \log n)$
- jeden z asymptotycznie najszybszych silnie wielomianowych algorytmów dla problemu najkrótszej ścieżki z wagami ≥ 0
- kopiec Fibonacciego to skomplikowana struktura – wymaga złożonej implementacji (w praktyce duży koszt)

AlSD – Wykład 20. Ścieżki w grafach

Notatki

Algorytm Dijkstry – uwagi i poprawność

Inne implementacje algorytmu Dijkstry

Istnieją implementacje algorytmu Dijkstry, które wykorzystują nieco inne podejście. Ich czas działania zależy od kosztów $c(u, v)$, ale dla pewnych rodzin grafów w praktyce okazują się być wydajne.

- algorytm Diala
- algorytm RADIX HEAP

Będą one omówione na kursie *Algorytmy optymalizacji dyskretnej*.

Poprawność algorytmu Dijkstry (formalnie)

Hipoteza indukcyjna (indukcja po $|S|$):

- dla $v \in S$ wartości $dist(v)$ są optymalne (permanentne)
- dla $v \in \bar{S}$ wartości $dist(v)$ są długościami najkrótszych ścieżek od s do v , których wszystkie węzły pośrednie są z S (przechodzą tylko przez węzły z optymalnymi wartościami $dist(v)$)

Hipoteza indukcyjna jest oczywista dla $S = \emptyset$.

AlSD – Wykład 20. Ścieżki w grafach

Notatki

Poprawność algorytmu Dijkstry

★ Szkic dowodu – część 1.

- w każdej iteracji do S dodajemy jeden wierzchołek $u \in \bar{S}$ t. że $dist(u) = \min\{dist(v) : v \in \bar{S}\}$
- z zał. ind. wiemy, że $dist(u)$ jest długością najkrótszej ścieżki P_0 od s do u przechodzącej jedynie po wierzchołkach z S
- rozważmy dowolną ścieżkę P' od s do u zawierającą wierzchołki pośrednie z \bar{S}
- niech w – pierwszy wierzchołek z \bar{S} na P'
 - w rozdziela P' na P'_1 i P'_2
 - P'_1 zawiera tylko wierzchołki pośrednie z S
- z zał. ind. i sposobu wyboru u mamy
$$length(P'_1) \geq dist(w) \geq dist(u) = \min\{dist(v) : v \in \bar{S}\}$$
($dist(w)$ to dl. najkrótszej ścieżki (po wierzch. z S) od s do w).
- $length(P'_2) \geq 0$, zatem $length(P') \geq dist(u) = length(P)$
- **Wniosek:** $dist(u)$ to długość najkrótszej ścieżki od s do u w grafie $G = (V, E)$

Notatki

Poprawność algorytmu Dijkstry

★ Szkic dowodu – część 2.

- po ustaleniu optymalnej wartości $dist(u)$ i przeniesieniu u z \bar{S} do S , długości najkrótszych ścieżek po wierzchołkach z S dla pozostałych wierzchołków z \bar{S} mogły zmaleć
 - ścieżki mogą teraz przechodzić dodatkowo przez u
- po oznaczeniu $dist(u)$ jako *permanentna*, algorytm wykonuje operację *distance update* dla wszystkich $v \in V$ t. że $(u, v) \in E$
- po tych update'ach ścieżka $s = v_0 - v_1 - \dots - v_h = v$ wyznaczona przez indeksy $pred(\cdot)$ spełnia
$$dist(v_{i+1}) = dist(v_i) + c(v_i, v_{i+1}), 0 \leq i < h$$
 - ta ścieżka po drodze przechodzi tylko po wierzchołkach z $S \cup \{u\}$
- można pokazać, że jeśli spełnione są powyższe równości, to wartości $dist(v)$ dla $v \in \bar{S} \setminus \{u\}$ są długościami najkrótszych ścieżek (przechodzących po węzłach z $S \cup \{u\}$)

Notatki

Literatura

Wykład 20. został w większości przygotowany na podstawie książki [DPV06] S. Dasgupta, Ch. Papadimitriou, U. Vazirani, *Algorithms*, 1st edition, McGraw-Hill Education, 2006 (fragmenty rozdziału 4)

Literatura dodatkowa

[AMO14] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Pearson Education, 2014 (fragmenty rozdziału 4)

[CLRS22] T.H. Cormen, Ch.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, 4th edition, MIT Press, 2022 (fragmenty rozdziałów 20 i 22)

Notatki

Notatki