Problem najkrótszych ścieżek z jednego źródła

Znajdowanie najkrótszych ścieżek pomiędzy węzłami grafu jest jednym z ważnych problemów praktycznych. Przykładowym zastosowaniem mogą być systemy nawigacji satelitarnej, które dla ustalonego położenia znajdują najkrótsze połączenie do innego położenia w sieci drogowej (lub rowerowej, itp.). Sieć dróg może być przedstawiona za pomocą grafu, którego węzłami są wszystkie połączenia lub skrzyżowania dróg, a łukami wszystkie odcinki dróg pomiędzy takimi połączeniami.

Jednak w praktyce każda nietrywialna sieć drogowa zawiera wiele tysięcy takich skrzyżowań i odcinków między nimi, i znalezienie najkrótszej ścieżki pomiędzy nawet niezbyt odległymi od siebie miejscami, które dzieli np. kilkanaście lub kilkadziesiąt skrzyżowań, może być nietrywialne, o ile algorytm będzie systematycznie analizował wszystkie możliwe ścieżki z pierwotnego położenia. Widać więc, że efektywny algorytm znajdowania takich połączeń byłby bardzo przydatny.

Definiujemy więc **problem najkrótszych ścieżek** w następujący sposób. Mamy dany graf skierowany $G = (V, E)$ z wagami określonymi funkcją $w : E \to \mathbb{R}$ przypisującą każdemu łukowi grafu **wagę**. Funkcja w jest następnie rozszerzona dla ścieżek w ten sposób, że dla każdej ścieżki $p = \langle v_0, v_1, ..., v_k \rangle$ jej waga jest sumą wag wszystkich łuków ścieżki:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Waga najkrótszej ścieżki $\delta(u, v)$ z węzła u do v jest określona przez:

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{jeśli istnieje ścieżka z } u \text{ do } v \\ \infty & \text{w przeciwnym wypadku} \end{cases}$$

Najkrótszą ścieżką z węzła u do v jest każda ścieżka p, której waga $w(p) = \delta(u, v)$. Celem jest znalezienie najkrótszej ścieżki w grafie z określonego wierzchołka do innego określonego wierzchołka.

Zauważmy, że poznany wcześniej algorytm przeszukiwania wszerz znajduje najkrótsze ścieżki z wybranego wierzchołka grafu bez wag. W takim grafie można uważać że wszystkie ścieżki posiadają identyczną jednostkową wagę. Poszukiwany algorytm będzie zatem jakby rozszerzeniem przeszukiwania wszerz, uwzględniającym zmienne wagi łuków.

Określenie „najkrótsza ścieżka" przypisuje wagom łuków znaczenie długości. Rzeczywiście, wagi połączeń w grafie mogą reprezentować odległości w sieci drogowej. Ale mogą również reprezentować inne metryki połączeń, których wartości dodają się na ścieżkach w grafie, takiej jak: czas przejazdu, zużycie paliwa, koszt przejazdu, itp.

Optymalna podstruktura problemu

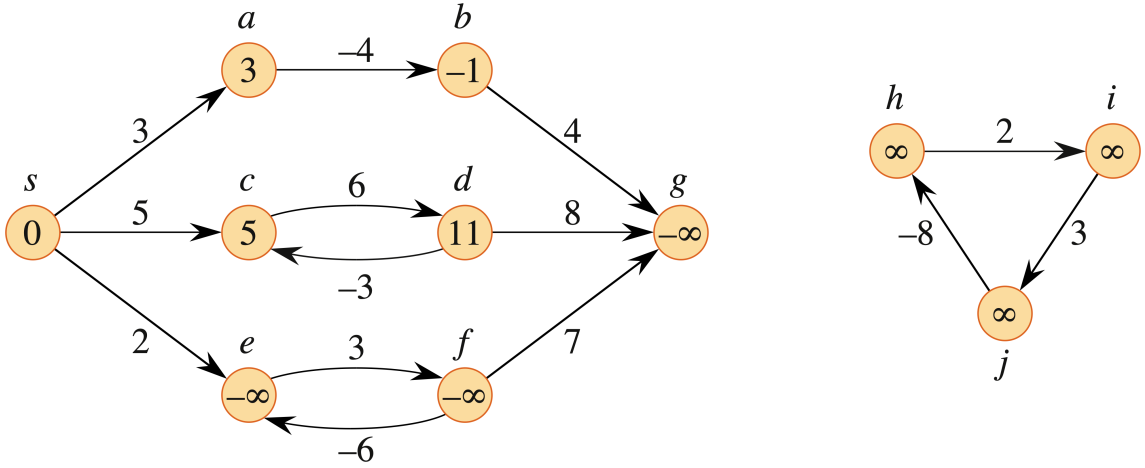
Można udowodnić, że jeśli pewna ścieżka $p = \langle v_0, v_1, ..., v_k \rangle$ jest najkrótszą ścieżką pomiędzy węzłami v_0 i v_k to dla każdej pary węzłów v_i, v_j na tej ścieżce, takimi, że $0 \leq i \leq j \leq k$, fragment ścieżki p pomiędzy tymi węzłami $p_{ij} = \langle v_i, v_{i+1}, ..., v_j \rangle$ jest najkrótszą ścieżką z v_i do v_j.

Mówiąc obrazowo, wszystkie fragmenty najkrótszej ścieżki są same w sobie również najkrótszymi ścieżkami.

Ta własność świadczy o **optymalnej substrukturze problemu** najkrótszych ścieżek, która była jedną z przesłanek do zastosowania programowania dynamicznego, i/lub algorytmów zachłannych. I rzeczywiście, oba te podejścia mają zastosowanie w znajdowaniu najkrótszych ścieżek. Algorytm Dijkstry przedstawiony poniżej jest algorytmem zachłannym.

Ujemne wagi połączeń

Jeśli w rozważanym grafie pewne ścieżki mają ujemne wagi to stanowi to pewien problem dla problemu najkrótszych ścieżek. Problemem są nie tyle indywidualne ujemne wagi, co możliwość istnienia cyklu o ujemnej wadze na jakiejkolwiek ścieżce z rozważanego wierzchołka. Jeśli pomiędzy dwoma wierzchołkami u, v istnieje ścieżka zawierająca taki cykl, to nie istnieje pomiędzy nimi ścieżka najkrótsza. Albowiem podążając tym cyklem wielokrotnie można dowolnie zmniejszać wagę ścieżki między nimi. Dla takich wierzchołków określamy $\delta(u, v) = -\infty$.



Problem widać na powyższym przykładowym grafie. Istnieją ścieżki z węzła s do węzłów e, f, g zawierające cykl o ujemnej wadze pomiędzy e i f. Dlatego mamy $\delta(s, e) = \delta(s, f) = \delta(s, g) = -\infty$. Jednocześnie, wierzchołki h, i, j są nieosiągalne z s, zatem zgodnie z definicją $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$, pomimo iż pomiędzy tymi węzłami istnieje cykl o ujemnej wadze.

Cykle w najkrótszych ścieżkach

Wiemy już, że najkrótsze ścieżki nie mogą zawierać cykli o wadze ujemnej. Pytanie jednak, czy mogą zawierać jakiekolwiek cykle.

Gdyby najkrótsza ścieżka pomiędzy pewnymi węzłami zawierała cykl o wadze dodatniej, to możnaby utworzyć nową poprawną ścieżkę między tymi samymi węzłami przez wycięcie z niej tego cyklu. Ale wtedy nowa ścieżka miałaby wagę mniejszą niż pierwotna, a zatem tamta nie mogłaby być najkrótsza.

Zatem żadna najkrótsza ścieżka na pewno nie zawiera cyklu o wadze dodatniej.

Pozostaje jednak możliwość istnienia w najkrótszej ścieżce cyklu o wadze równej zero. Po wycięciu tego cyklu ze ścieżki, nowa ścieżka również byłaby poprawną ścieżką pomiędzy tymi samymi węzłami, i tak samo jak pierwotna ścieżka byłaby najkrótsza.

Dla uproszczenia, i wprowadzenia pewnej jednoznaczności, będziemy zakładać, że najkrótsze ścieżki nie zawierają cykli.

Reprezentacja najkrótszych ścieżek

Rozwiązaniem problemu najkrótszych ścieżek powinien być rozkład długości najkrótszych ścieżek z określonego wierzchołka do innego wybranego wierzchołka (lub wybranych wierzchołków). Algorytmy przedstawione w tej prezentacji obliczają te długości dla wszystkich pozostałych wierzchołków grafu.

Jednak same długości ścieżek są niewystarczające. Dla wielu zastosowań potrzebne jest również efektywne wyznaczanie tych najkrótszych ścieżek. Dla ścieżki do jednego wierzchołka docelowego wynikiem pracy algorytmu mogłaby być sekwencja łuków (lub wierzchołków) najkrótszej ścieżki ze źródła do celu. Jednak jeśli wynikiem mają być najkrótsze ścieżki do wielu wierzchołków, to ta postać rozwiązania jest niepraktyczna.

Będziemy stosować rozwiązanie analogiczne do tego wykorzystanego w algorytmie przeszukiwania wszerz, to znaczy tworzenia w każdym węźle v atrybutu $v.\pi$ stanowiącego wskaźnik (identyfikator) poprzednika węzła v na najkrótszej ścieżce z globalnego źródła.

Atrybuty π są inicjalizowane wartością NIL, a następnie w trakcie pracy algorytmu przyjmują wartości innych węzłów grafu. W trakcie pracy algorytmu (każdego z dwóch przedstawionych poniżej) ustawiona wartość atrybutu π wskazuje na poprzednik węzła na jakiejś ścieżce od źródła do tego węzła. Jednak do momentu zakończenia algorytmu może nie być to ścieżka najkrótsza.

Atrybuty π tworzone przez algorytm generują **graf poprzedników** $G_\pi = (V_\pi, E_\pi)$:

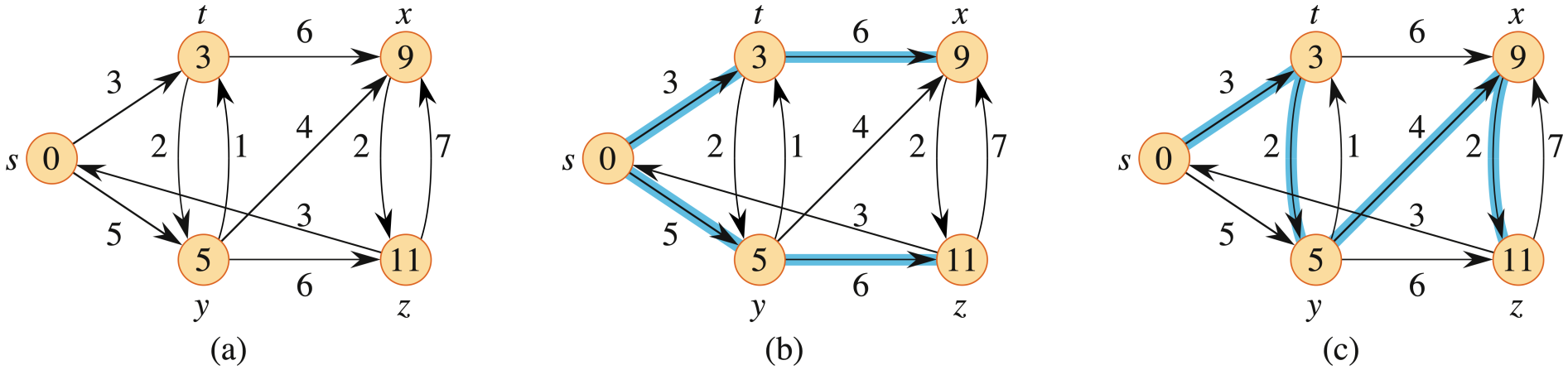
$$\begin{aligned} V_\pi &= \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\} \ , \\ E_\pi &= \{(v.\pi, v) \in E : v \in V_\pi - \{s\}\} \ . \end{aligned}$$

Drzewa przeszukiwania wszerz poznane wcześniej stanowiły również grafy poprzedników. Analogicznie do tych drzew zdefiniujemy obecnie **drzewo najkrótszych ścieżek** dla skierowanego grafu $G = (V, E)$ i korzeniu s jako skierowany graf $G' = (V', E')$, gdzie $V' \in V$ i $E' \in E$, takie że:

1. V' jest zbiorem wierzchołków w G osiągalnych z s,
2. G' jest drzewem z korzeniem w s,
3. dla każdego wierzchołka $v \in V'$ jedyna prosta ścieżka z s do v w G' jest najkrótszą ścieżką z s do v w G..

Drzewa najkrótszych ścieżek nie muszą być unikalne, podobnie jak najkrótsze ścieżki. Dla danego grafu i wybranego źródła może istnieć jedno lub więcej takich drzew.

Poniższy rysunek przedstawia przykładowy skierowany graf z wagami (a), w którym liczby w węzłach reprezentują wagę najkrótszej ścieżki z s do danego węzła. Rysunki (b) i (c) przedstawiają kolorem niebieskim dwa różne drzewa najkrótszych ścieżek dla tego grafu ze źródłem s:



Relaksacja

Przed przedstawieniem konkretnych algorytmów warto wprowadzić pewne elementarne procedury w nich wykorzystywane.

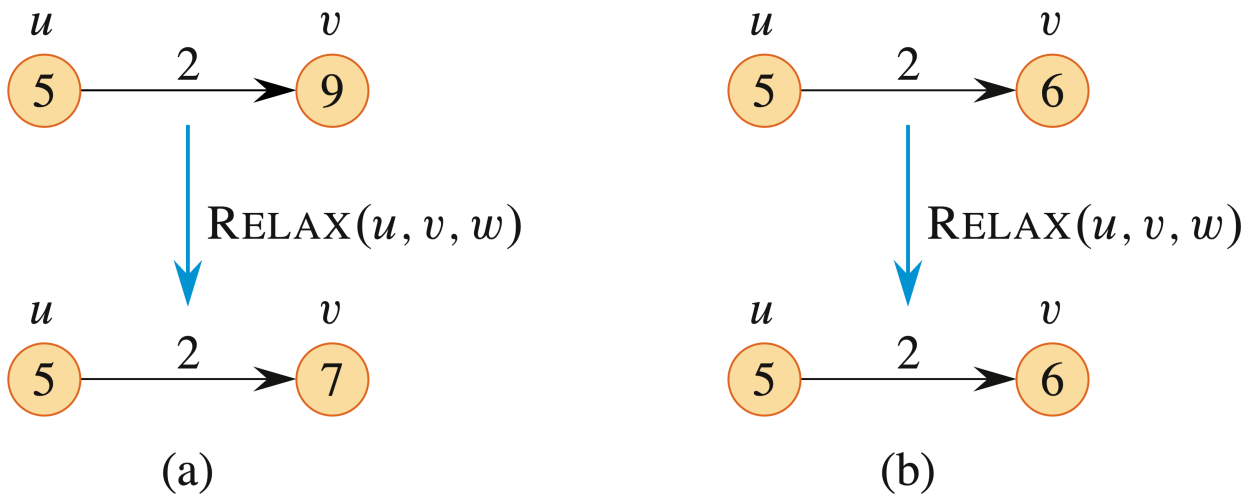
Poza atrybutem π reprezentującym poprzednika będziemy wykorzystywać atrybut d reprezentujący długość najkrótszej ścieżki z wybranego źródła do danego węzła. Podobnie jak wartość atrybutu π, w czasie pracy algorytmu wartość tego atrybutu może nie być równa długości najkrótszej ścieżki z s, lecz zawsze będzie jej ograniczeniem górnym. Oba atrybuty należy zainicjalizować za pomocą poniższej procedury INITIALIZE-SINGLE-SOURCE:

INITIALIZE-SINGLE-SOURCE(G, s)

1 **for** each vertex $v \in G.V$
2 $\quad v.d = \infty$
3 $\quad v.\pi = \text{NIL}$
4 $s.d = 0$

Procedura **relaksacji** krawędzi skierowanej (u, v) polegać będzie na sprawdzeniu, czy najkrótsza ścieżka z globalnego źródła s do wierzchołka v nie byłaby krótsza, gdyby prowadziła przez tę krawędź.

Ilustruje to poniższy rysunek. W przypadku (a) widać, że mając najkrótszą ścieżkę z s do u o długości 5, i aktualnie najkrótszą ścieżkę z s do v o długości 9, można uzyskać krótszą ścieżkę z s do v prowadzącą przez wierzchołek u i relaksowaną krawędź. Nowa najkrótsza ścieżka z s do v będzie miała długość 7.



Jednocześnie na rysunku (b) widać, że procedura relaksacji krawędzi (u, v) w tym przypadku nic nie zmieni, ponieważ ścieżka z s do v biegnąca przez wierzchołek u nie będzie krótsza niż najkrótsza znana obecnie ścieżka o długości 6.

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2        v.d = u.d + w(u, v)
3        v.π = u
```

Oba algorytmy przedstawione poniżej najpierw inicjalizują wymienione atrybuty za pomocą powyższej procedury INITIALIZE-SINGLE-SOURCE, a następnie powtarzalnie aktualizują je za pomocą procedury relaksacji.

Własności najkrótszych ścieżek i relaksacji

Poniższe własności zakładają, że graf został zainicjalizowany procedurą INITIALIZE-SINGLE-SOURCE, oraz, że wszelkie zmiany atrybutów π i d nastąpiły wskutek zastosowania sekwencji kroków relaksacji.

Nierówność trójkąta

Dla każdej krawędzi $(u, v) \in E$ zachodzi $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Własność górnego ograniczenia

Dla wszystkich wierzchołków $v \in V$ zawsze zachodzi $v.d \geq \delta(s, v)$.
Ponadto, od momentu gdy $v.d$ osiąga wartość $\delta(s, v)$, nie ulega już zmianie.

Własność braku ścieżki

Jeśli nie istnieje ścieżka z s do v, to zawsze zachodzi $v.d = \delta(s, v) = \infty$.

Własność zbieżności

Jeśli dla pewnych wierzchołków $u, v \in V$ $p = s \rightsquigarrow u \rightarrow v$ jest najkrótszą ścieżką w grafie $G = (V, E)$, i jeśli w dowolnej chwili przed relaksacją krawędzi (u, v) zachodzi $u.d = \delta(s, u)$, to po jej relaksacji już zawsze będzie $v.d = \delta(s, v)$.

Własność relaksacji dla ścieżki

Jeśli $p = \langle v_o, v_1, ..., v_k \rangle$ jest najkrótszą ścieżką z $s = v_0$ do v_k, i krawędzie ścieżki p są relaksowane w kolejności $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$, to $v_k.d = \delta(s, v_k)$.
Ta własność zachodzi niezależnie od innych kroków relaksacji, nawet przemieszanych z relaksacją krawędzi ścieżki p.

Własność podgrafu poprzedników

Od momentu gdy $v.d = \delta(s, v)$ dla wszystkich $v \in V$, podgraf poprzedników jest drzewem najkrótszych ścieżek z korzeniem w s.

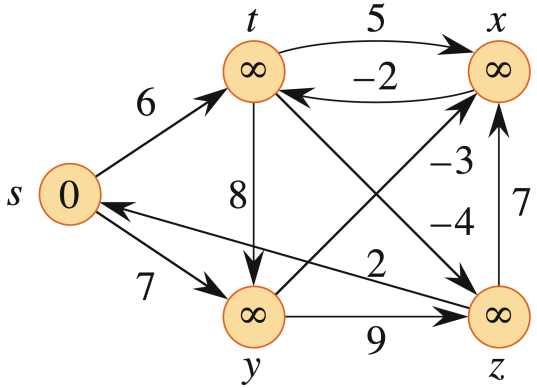
Algorytm Bellmana-Forda

Algorytm Bellmana-Forda rozwiązuje problem najkrótszych ścieżek z jednego źródła. Algorytm działa w przypadku ogólnym, to znaczy również dla grafu z ujemnymi wagami ścieżek. Jednocześnie algorytm sprawdza, czy ze źródła nie jest osiągalna jakaś ścieżka zawierająca cykl o ujemnej wadze. Jeśli taka ścieżka istnieje, to nie jest możliwe wyznaczenie wszystkich najkrótszych ścieżek. Gdy takiej ścieżki nie ma, to algorytm oblicza wszystkie najkrótsze ścieżki i ich wagi.

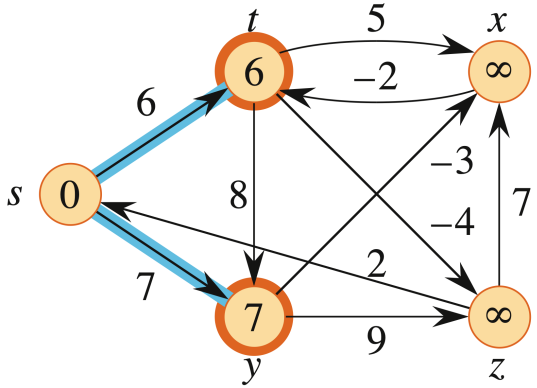
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3        for each edge (u, v) ∈ G.E
4              RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6        if v.d > u.d + w(u, v)
7              return FALSE
8  return TRUE
```

Algorytm relaksuje krawędzie, zmniejszając stopniowo oszacowanie $v.d$ najkrótszej ścieżki z s do każdego $v \in V$, aż zostanie osiągnięta waga najkrótszej ścieżki $\delta(s, v)$.

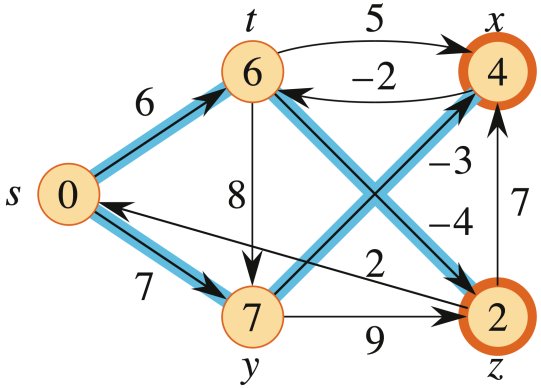
Przykład wykonania algorytmu Bellmana-Forda na grafie ze źródłem s (a). Rysunki (b)–(e) przedstawiają stan grafu po każdorazowym wykonaniu pętli **for** z wierszy 2–4 algorytmu. Krawędzie są przetwarzane w kolejności: $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$. Węzły zawierają aktualne wartości d a niebieskie krawędzie wskazują aktualne indeksy poprzedników π. Pomarańczowa obwódka wskazuje węzły, których atrybuty uległy zmianie w wyniku relaksacji w danym kroku.



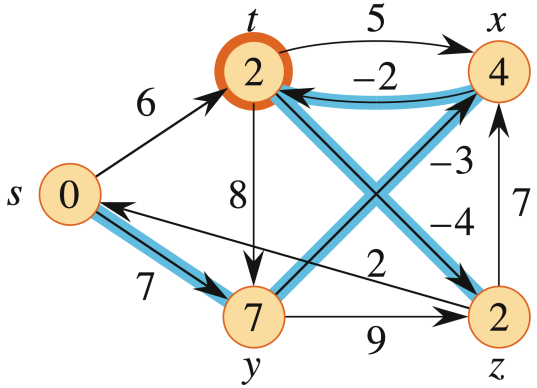
(a)

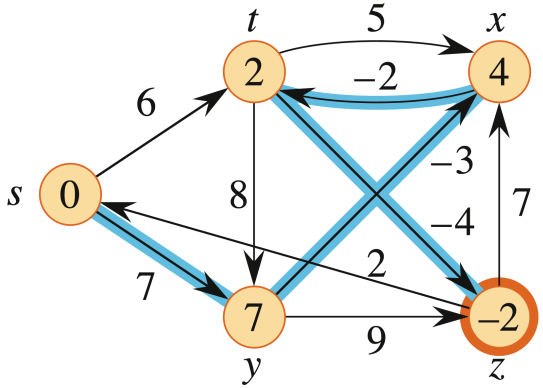


(b)



(c)





Algorytm Bellmana-Forda jest niedeterministyczny, ponieważ pętla **for** w wierszu 3 wybiera łuki w nieokreślonej kolejności. W rezultacie różne wywołania algorytmu mogą wygenerować różne rozwiązania.

Poprawność algorytmu Bellmana-Forda jest nieoczywista, a jej dowód jest nietrywialny. Wykorzystuje on między innymi własność, że każda niezawierająca cyklu ścieżka zawiera co najwyżej $|V| - 1$ krawędzi, oraz wymienione wcześniej własności relaksacji dla ścieżki, i podgrafu poprzedników.

Czas wykonania algorytmu Bellmana-Forda wynosi $O(V^2 + VE)$ dla grafów zapisanych za pomocą list sąsiedztwa. Niekiedy wystarczających jest mniej niż $|V| - 1$ przebiegów pętli z kroków 2–4, co jest przyczyną zapisu $O(V^2 + VE)$ zamiast $\Theta(V^2 + VE)$.

W typowym przypadku gdy $|E| = \Omega(V)$, można ten czas wykonania zapisać jako $O(VE)$.

Algorytm Dijkstry

Algorytm Dijkstry również rozwiązuje problem najkrótszych ścieżek z jednego źródła, ale zakłada nieujemność wag wszystkich krawędzi. Za to, przy dobrej implementacji, algorytm Dijkstry ma czas działania lepszy niż algorytm Bellmana-Forda.

Algorytm Dijkstry można uważać za uogólnienie przeszukiwania wszerz na grafy z wagami. W tamtym algorytmie kolejka chronologiczna FIFO była stosowana do zapewnienia, że dłuższe ścieżki będą sprawdzane po tym jak sprawdzone zostaną, wcześniej umieszczone w kolejce, ścieżki krótsze.

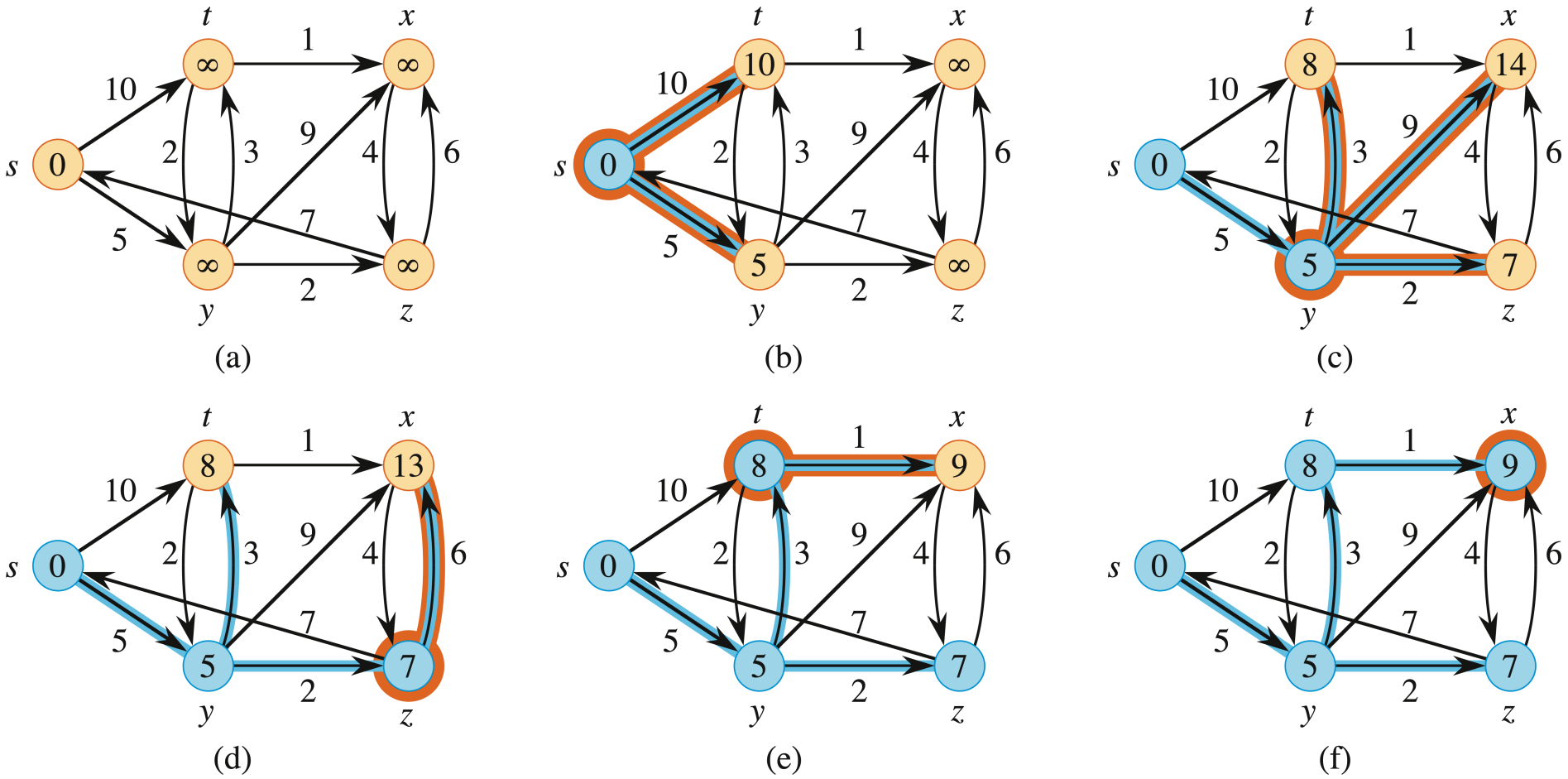
Obecnie nie możemy tak zrobić, ponieważ wszystkie ścieżki mają dowolnie wybrane wagi, i jest możliwe, że później znaleziona ścieżka będzie miała krótszą odległość od źródła. Dlatego zamiast kolejki chronologicznej FIFO zastosujemy obecnie kolejkę priorytetową porządkowaną według znanej minimalnej odległości od źródła. W każdym kroku algorytmu do analizy będzie wybierany węzeł o minimalnej odległości od źródła.

Podobnie jak zakres węzłów objętych przeszukiwaniem wszerz „rozlewa się" po grafie jednostkowymi krokami, w algorytmie Dijkstry ten zakres rozszerza się wzdłuż najkrótszych ścieżek, nigdy nie przekraczając węzła o danej szacowanej odległości od źródła, gdy nie są sprawdzone łuki z innego węzła o mniejszej szacowanej odległości od źródła.

```
DIJKSTRA(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S = ∅
 3  Q = ∅
 4  for each vertex u ∈ G.V
 5        INSERT(Q, u)
 6  while Q ≠ ∅
 7        u = EXTRACT-MIN(Q)
 8        S = S ∪ {u}
 9        for each vertex v ∈ G.Adj[u]
10              RELAX(u, v, w)
11              if the call of RELAX decreased v.d
12                    DECREASE-KEY(Q,v,v.d)
```

Algorytm jest zachłanny, ponieważ w każdym przebiegu pętli **while** w wierszach 6—12 wybiera węzeł o najkrótszej odległości od źródła, i relaksuje łuki do jego sąsiadów, być może aktualizując ich odległość od źródła. Zbiór $S = V - Q$ zawiera wszystkie węzły u, którym zostały już *zrelaksowane* krawędzie z u do wszystkich ich sąsiadów. Można udowodnić, że oszacowanie $u.d = \delta(s, u)$ już w momencie dodawania węzła u do zbioru S.

Przykład analizy algorytmem Dijkstry grafu na rysunku (a) ze źródłem s. Węzły zawierają oszacowanie najmniejszej odległości od źródła. Węzły koloru piaskowego należą do kolejki Q i w kolejnych iteracjach pętli **while** na rysunkach (b)–(f) są przenoszone do zbioru S stając się niebieskimi. Niebieskie łuki wyznaczają poprzedniki węzłów. Węzły podświetlone na pomarańczowo oznaczają wybrane minima z kolejki, a łuki podświetlone na pomarańczowo wskazują że wartość d i wskaźnik poprzednika π uległy zmianie w następniku tego łuku.



Podobnie jak algorytm Bellmana-Forda, algorytm Dijkstry jest niedeterministyczny, i jego różne wywołania mogą wygenerować różne rozwiązania. Źródłem niedeterminizmu jest zarówno wybór wierzchołków do relaksacji w pętli **for** w wierszu 9 algorytmu, jak również wybór wierzchołka przenoszonego z kolejki Q do zbioru S, o ile istnieją w tej kolejce różne węzły o identycznym oszacowaniu d odległości od źródła.

Oszacowanie czasu wykonania algorytmu Dijkstry zależy od implementacji operacji na kolejce Q. Przy prostej implementacji za pomocą zwykłej tablicy, czas wykonania algorytmu wynosi $O(V^2 + E)$ co dzięki własności $E = O(V^2)$ można przepisać jako $O(V^2)$.

Jednak przy zastosowaniu stogu Fibonacciego do implementacji kolejki priorytetowej, czas wykonanie poprawia się do $O(V \log V + E)$.

Znajdowanie wszystkich ścieżek z jednego źródła

W kontekście przedstawionego zastosowania w systemach nawigacji, i motywacji, że poszukujemy algorytmu bardziej efektywnego, niż znajdowanie poszukiwanej trasy przez próbowanie wszystkich możliwości, może wydawać się dziwne, że oba przedstawione algorytmy znajdują najkrótsze ścieżki do wszystkich węzłów w grafie.

Czy nie byłoby prościej znaleźć tylko ścieżkę do jednego wybranego węzła docelowego? Czy nie mogłoby to być bardziej efektywne?
Na przykład, mając mapę wszystkich połączeń drogowych w Europie, i chcąc znaleźć najkrótszą drogę z Kutna do Zduńskiej Woli, czy nie ma prostszego sposobu?

Odpowiedź brzmi, że owszem, w konkretnym przypadku możemy zmodyfikować algorytm Dijkstry aby zatrzymał się wcześniej (dokładnie: w momencie przenoszenia Zduńskiej Woli do zbioru S), bez obliczania wszystkich najkrótszych ścieżek z Kutna do wszystkich miejsc w Europie. Jednak trudno udowodnić, że algorytm Dijkstry z tą modyfikacją będzie asymptotycznie szybszy niż w ogólności.

Albowiem algorytm relaksuje ścieżki, tzn. po znalezieniu jednej potencjalnie najkrótszej ścieżki do danego węzła, sprawdza, czy nie ma innej ścieżki jeszcze krótszej. Tę procedurę relaksacji można bezpiecznie zatrzymać dopiero wtedy, gdy mamy pewność, że nie znajdzie się już żadna inna krótsza ścieżka do naszego celu.

W przypadku algorytmu Bellmana-Forda sytuacja jest jeszcze gorsza, bo dopuszcza on istnienie łuków z ujemnymi wagami. Mówiąc obrazowo, po sprawdzeniu, na przykład, wszystkich ścieżek z Kutna do Zduńskiej Woli w obrębie Polski, i mając aktualnie najkrótszą ścieżkę, nadal nie mamy gwarancji, że nie istnieje ścieżka prowadząca np. przez Rzym, która dzięki odcinkowi o ujemnej wadze jest w stanie skrócić tę podróż.

A jak radzą sobie z tym prawdziwe systemy nawigacji?

Po pierwsze, nie biorą one pod uwagę wag ujemnych, a zatem punktem wyjścia jest dla nich algorytm Dijkstry. Stosując opisany warunek wczesnego zatrzymania algorytmu możemy uniknąć przeszukiwania całej Europy, ograniczając się do wszystkich ścieżek długości około 100 kilometrów od Kutna (w tej odległości znajduje się Zduńska Wola).

Dodatkowo, systemy te mogą stosować pewne techniki z dziedziny sztucznej inteligencji, i jeszcze bardziej usprawnić ten proces. Wkrótce przyjrzymy się tym metodom.

Krótkie podsumowanie — pytania sprawdzające

1. Znajdź dwa drzewa najkrótszych ścieżek dla grafu z rysunku na stronie 9 inne niż te pokazane na rysunkach.

2. Wykonaj algorytm Bellmana-Forda dla grafu z rysunku na stronie 18, przyjmując wierzchołek z za źródło. W każdym przebiegu wykonaj relaksacje krawędzi w tym samym porządku jaki został podany. Podaj wszystkie wartości d i π po każdym przebiegu.
   Następnie zmień wagę krawędzi (z, x) na 4 i wykonaj algorytm ponownie, przyjmując tym razem za źródło wierzchołek s.

3. Wykonaj algorytm Dijkstry dla grafu z rysunku na str.9, najpierw przyjmując za źródło wierzchołek s, a następnie wierzchołek z. W ten sam sposób jak na rysunku na stronie 23 pokaż wartości d i π, oraz wierzchołki zbioru S po każdej iteracji pętli **while**.

Literatura i materiały pomocnicze

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest, Clifford Stein: Wprowadzenie do algorytmów, PWN, 2024, rozdział 22.