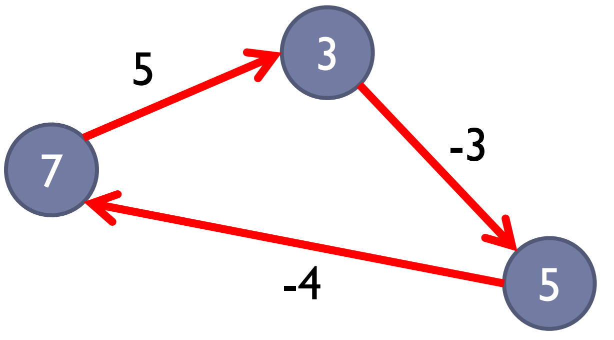


Digrafy, silnie spójne składowe, cykliczność najkrótsze drogi w sieciach

dr Anna Beata Kwiatkowska, UMK Toruń

Silna spójność digrafów

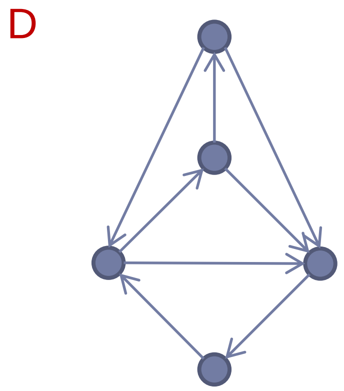
Definicja 6.1

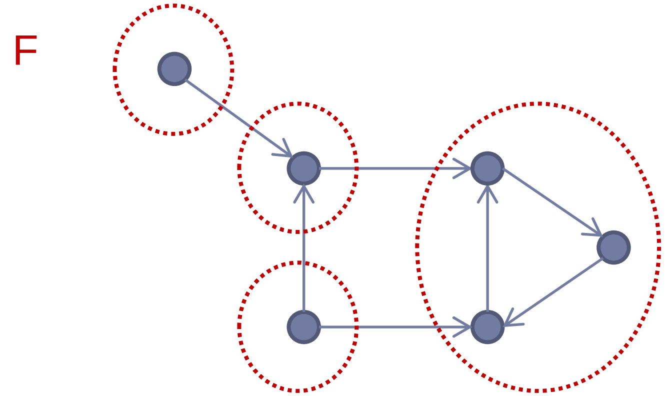
Digraf D jest silnie spójny jeśli istnieje droga zorientowana od dowolnego wierzchołka do każdego innego wierzchołka.

Digraf D jest słabo spójny, jeśli nie jest silnie spójny, natomiast spójny jest graf G, który powstaje z digrafu D po pominięciu zorientowania jego łuków.

Silnie spójna składowa digrafu D to jego maksymalny silnie spójny poddigraf.

Przykład: Graf D silnie spójny i F słabo spójny. F ma cztery silnie spójne składowe.



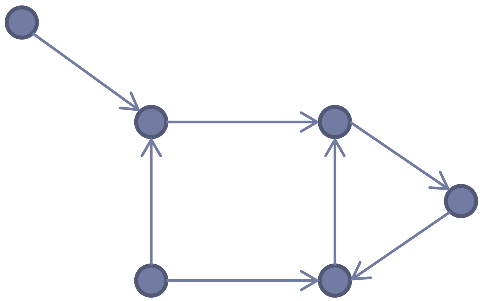


Silnie spójne składowe - algorytm

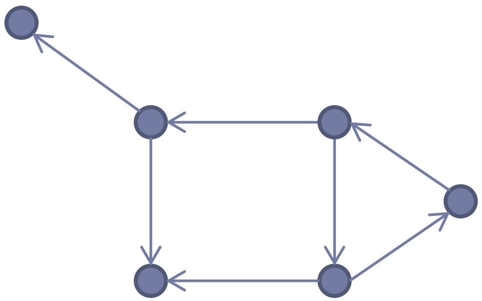
O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.

D



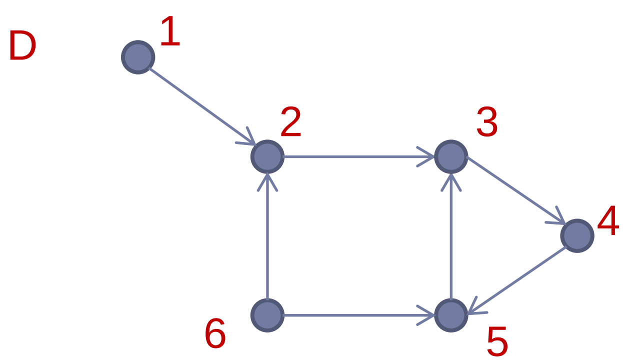
D^T

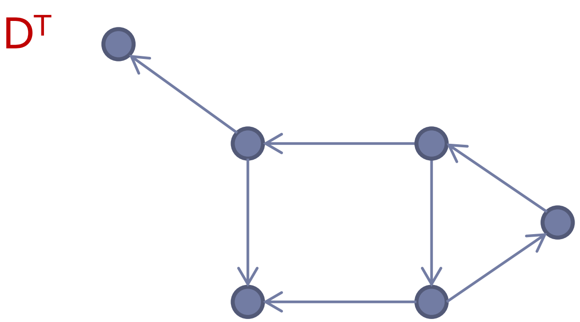


Silnie spójne składowe - algorytm

O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.
2. Wyznaczamy kolejność postorder wierzchołków digrafu D algorytmem przeszukiwania w głąb (DFS). Wywołujemy go tyle razy, aż odwiedzimy wszystkie wierzchołki digrafu D.

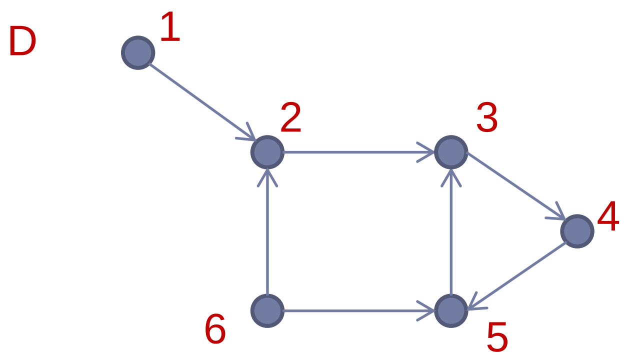


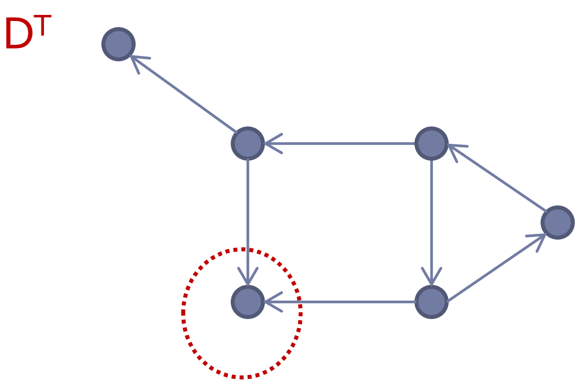


Silnie spójne składowe - algorytm

O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.
2. Wyznaczamy kolejność postorder wierzchołków digrafu D algorytmem przeszukiwania w głąb (DFS). Wywołujemy go tyle razy, aż odwiedzimy wszystkie wierzchołki digrafu D.
3. Wywołujemy algorytm DFS dla grafu transponowanego D^T od wierzchołków wziętych w kolejności odwrotnej niż postorder. Wszystkie wierzchołki osiągnięte w kolejnych przeszukiwaniach w głąb należą do jednej silnie spójnej składowej.

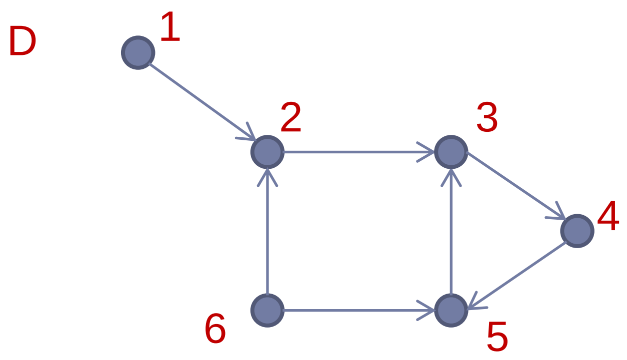


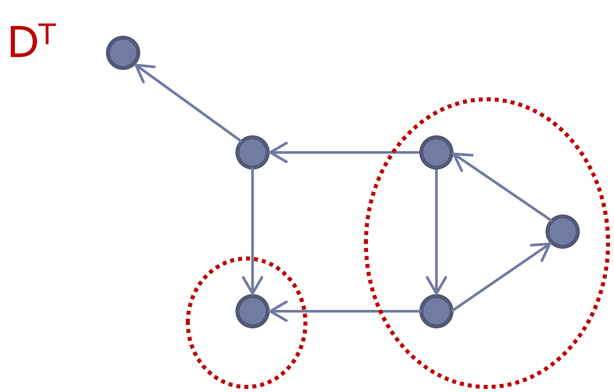


Silnie spójne składowe - algorytm

O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.
2. Wyznaczamy kolejność postorder wierzchołków digrafu D algorytmem przeszukiwania w głąb (DFS). Wywołujemy go tyle razy, aż odwiedzimy wszystkie wierzchołki digrafu D.
3. Wywołujemy algorytm DFS dla grafu transponowanego D^T od wierzchołków wziętych w kolejności odwrotnej niż postorder. Wszystkie wierzchołki osiągnięte w kolejnych przeszukiwaniach w głąb należą do jednej silnie spójnej składowej.

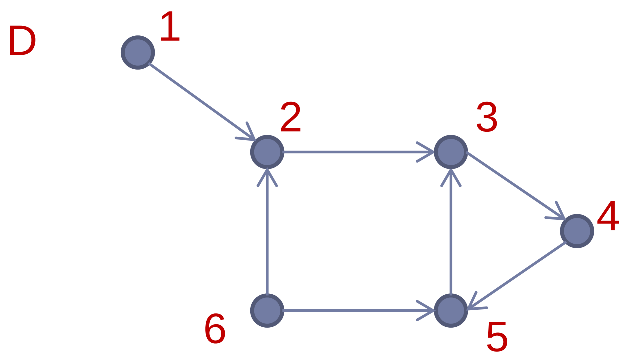


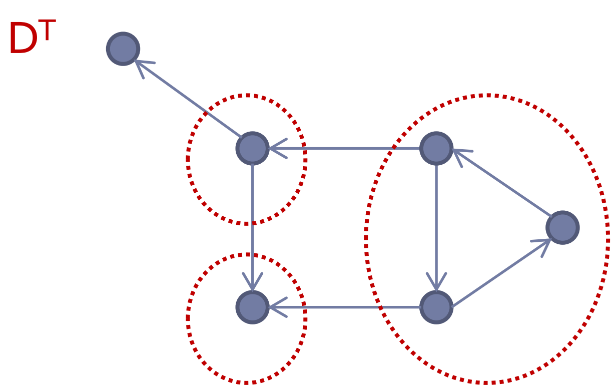


Silnie spójne składowe - algorytm

O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.
2. Wyznaczamy kolejność postorder wierzchołków digrafu D algorytmem przeszukiwania w głąb (DFS). Wywołujemy go tyle razy, aż odwiedzimy wszystkie wierzchołki digrafu D.
3. Wywołujemy algorytm DFS dla grafu transponowanego D^T od wierzchołków wziętych w kolejności odwrotnej niż postorder. Wszystkie wierzchołki osiągnięte w kolejnych przeszukiwaniach w głąb należą do jednej silnie spójnej składowej.

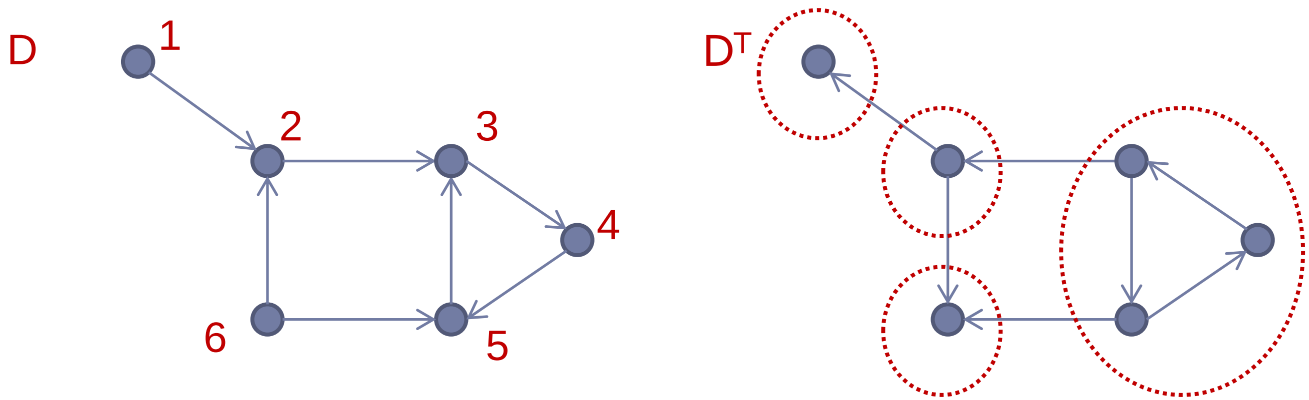




Silnie spójne składowe - algorytm

O(n+m)

1. Rozpatrujemy digraf D oraz digraf transponowany D^T, który powstaje przez odwrócenie łuków zwrotami. W implementacji wczytując dane od razu tworzymy reprezentacje obydwu digrafów w postaci list sąsiedztwa.
2. Wyznaczamy kolejność postorder wierzchołków digrafu D algorytmem przeszukiwania w głąb (DFS). Wywołujemy go tyle razy, aż odwiedzimy wszystkie wierzchołki digrafu D.
3. Wywołujemy algorytm DFS dla grafu transponowanego D^T od wierzchołków wziętych w kolejności odwrotnej niż postorder. Wszystkie wierzchołki osiągnięte w kolejnych przeszukiwaniach w głąb należą do jednej silnie spójnej składowej.



Digrafy acykliczne

Definicja 6.2

Digraf acykliczny (DAG – *ang. directed acyclic graph*) to graf niezwierający cykli zorientowanych, tzn. poruszając się od dowolnego wierzchołka zgodnie z kierunkiem łuków, nigdy nie powrócimy do tego samego wierzchołka.

Źródłem nazywamy wierzchołek do którego nie wchodzi żaden łuk. Ujściem nazywamy wierzchołek z którego nie wychodzi żaden łuk.

Jest to bardzo ważna klasa grafów. Zastosowania:

- w obliczeniach numerycznych,
- przetwarzaniu potokowym (np. projektowaniu procesorów),
- tworzeniu układów kombinacyjnych z bramek logicznych,
- kompresji,
- kompilatorach,
- diagramach przepływu,
- przepływie danych czy zasobów (transport, energia elektryczna, woda),
- porządek topologiczny (patrz poprzednie wykłady)

Digrafy acykliczne

Twierdzenie 6.1

Jeśli digraf $D=(V,U)$ jest acykliczny to ma źródło i ujście.

Dowód:

Załóżmy, że D nie ma ujścia (analogicznie dowodzimy dla źródła).

Wybierzmy zatem dowolny wierzchołek $x_0 \in V$. Nie jest on ujściem, dlatego istnieje łuk w U skierowany od x_0 do innego wierzchołka, np. do x_1 który też nie jest ujściem.

Powtarzamy rozumowanie dla x_1 i osiągamy wierzchołek x_2, itd. Otrzymujemy nieskończony ciąg wierzchołków $x_0, x_1, x_2, \ldots$ dla skończonej liczby wierzchołków, musi zatem on zawierać powtórzenia.

Niech i, j, gdzie $i<j$ będą miejscami, na których występuje ten sam wierzchołek, a nie występuje między tymi miejscami.

Wierzchołki na miejscach $i, i+1, \ldots j$ definiują cykl w D.

Otrzymujemy sprzeczność. □

Sieci

Definicja 6.3

W grafie G (V,E) (lub digrafie) definiujemy funkcją na jego krawędziach (łukach), określającą wagę każdej z nich:

$$w : E \longrightarrow R$$

Graf (digraf) G z tak określoną funkcją wag nazywamy grafem ważonym (siecią).

Definicja 6.4.

Dla podgrafu H (V_H, E_H) grafu G określamy długość d podgrafu H, jako sumę wag wszystkich jego krawędzi e:

$$d(H) = \sum_{e \in E_H} w(e)$$

Mając tak określoną funkcję d możemy mówić np. o długości drogi w grafie, o długości drzewa itd.

Przykładowa sieć

Dane

n= 8, m=11

1 2 2

1 3 3

1 4 4

1 6 5

3 7 5

3 5 7

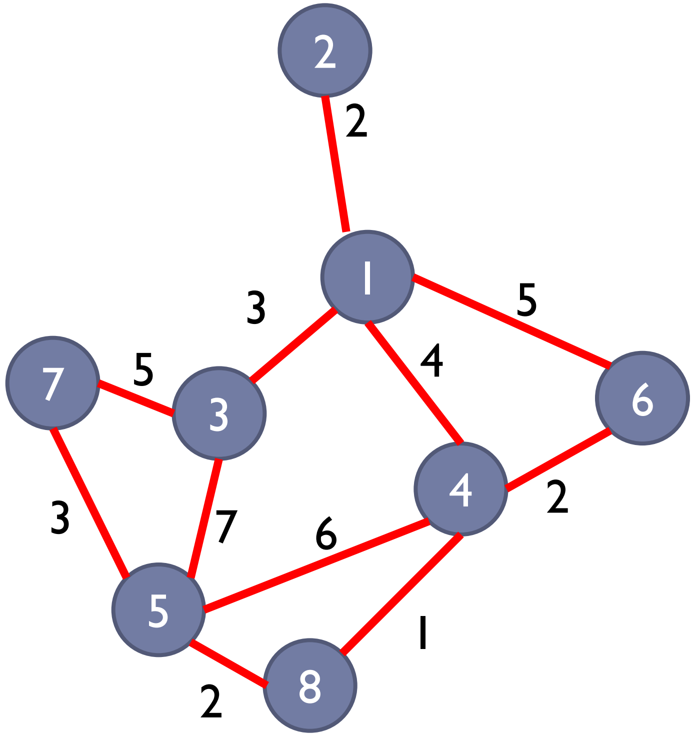
4 5 6

4 8 1

4 6 2

5 7 3

5 8 2



Trzecia liczba w danych dla krawędzi oznacza wagę danej krawędzi.

Reprezentacja sieci w komputerze

Niech dana będzie sieć G (V,E), gdzie |V|=n, |E|=m z funkcją wag w określona dla każdej krawędzi (łuku).

Reprezentowanie sieci w komputerze może być realizowana na różne sposoby:

- macierz sąsiedztwa W o rozmiarze n x n z wartościami wag, W[u, v]=w(e) gdzie wierzchołki u, v wyznaczają krawędź (łuk) e – potrzebna pamięć n^2
- lista sąsiadów oraz oddzielnie macierz – potrzebna pamięć to $2m+n^2$ (dla łuków $m+n^2$)
- Rozbudowana lista sąsiadów – zamiast podwieszać sąsiada, podwieszamy parę: waga krawędzi i sąsiad, do którego ta krawędź prowadzi – potrzebna pamięć 4m (dla łuków 2m)

Struktury dla naiwnej implementacji w Pythonie:

- lista list dla macierzy sąsiedztwa z wagami
- słownik list dla listy sąsiadów grafu
- listy dla odległości, ustalonych najkrótszych odległości, ojców

Problem najkrótszych dróg - ogólnie

Dane:

Graf G=(V, E) (lub digraf)

V – niepusty zbiór wierzchołków

E – zbiór krawędzi (łuków)

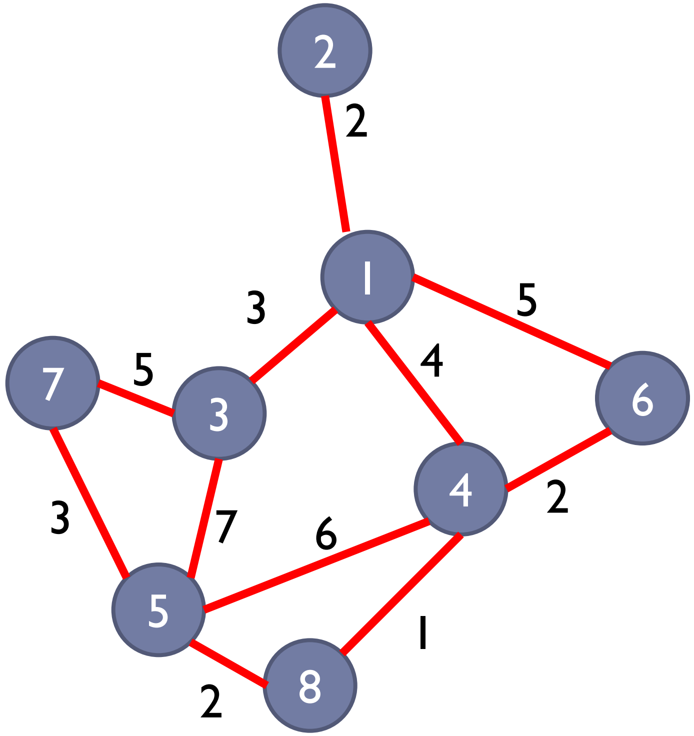
$w : E \rightarrow R$ - funkcja wag

v, u wierzchołki z V

Wynik:

Najkrótsza droga z v do u

w sieci G.



Przykład:

Przeanalizuj rysunek sieci.

Jaka jest najkrótsza droga z wierzchołka numer 2 do wierzchołka o numerze 5?

Najkrótsze drogi - problemy

Rodzaje problemów:

- najkrótsza (najszybsza, najtańsza itp.) droga w sieci z ustalonego wierzchołka do wszystkich pozostałych wierzchołków lub między wszystkimi parami wierzchołków,
- najkrótsza droga z jednego do innego wierzchołka przechodząca przez wyszczególnione wierzchołki,
- najkrótsza, druga najkrótsza,…, wszystkie najkrótsze drogi,
- itd.

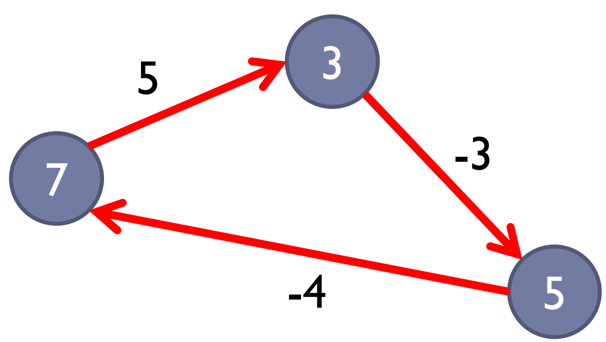
Problemy pokrewne:

- najdłuższa droga,
- najbardziej niezawodna droga,
- droga o największej przepustowości,
- wyznaczanie tras objazdu.

Najkrótsze drogi - algorytmy

- Znajdowanie najkrótszej drogi z ustalonego wierzchołka początkowego s (źródła dla drzewa najkrótszych dróg) do innego ustalonego wierzchołka t (ujścia dla drzewa najkrótszych dóg) w sieci skierowanej o **nieujemnych** wagach łuków – algorytm Dijkstry (1959 r) – metoda ustalania cech.
- Znajdowanie najkrótszej drogi z ustalonego wierzchołka początkowego s (źródła) do innego ustalonego wierzchołka t (ujścia) w sieci skierowanej o **dowolnych** wagach łuków – algorytm Moore'a (1957)–Bellmana (1958) – metoda poprawiania cech.
- Wyznaczanie najkrótszych dróg **między każdą parą wierzchołków** w sieci skierowanej o dowolnych wagach łuków, nie zwierającej cykli ujemnej długości – algorytm Floyda (1962)-Warshalla (1962)

Przykład Cykl o ujemnej długości – nie można znaleźć w nim najkrótszej drogi



Problem najkrótszych dróg – algorytm Dijkstry

Dane:

G=(V, E)

V – niepusty zbiór wierzchołków

E – zbiór krawędzi

w : E $\longrightarrow$ $R_{\geq 0}$ - funkcja wag

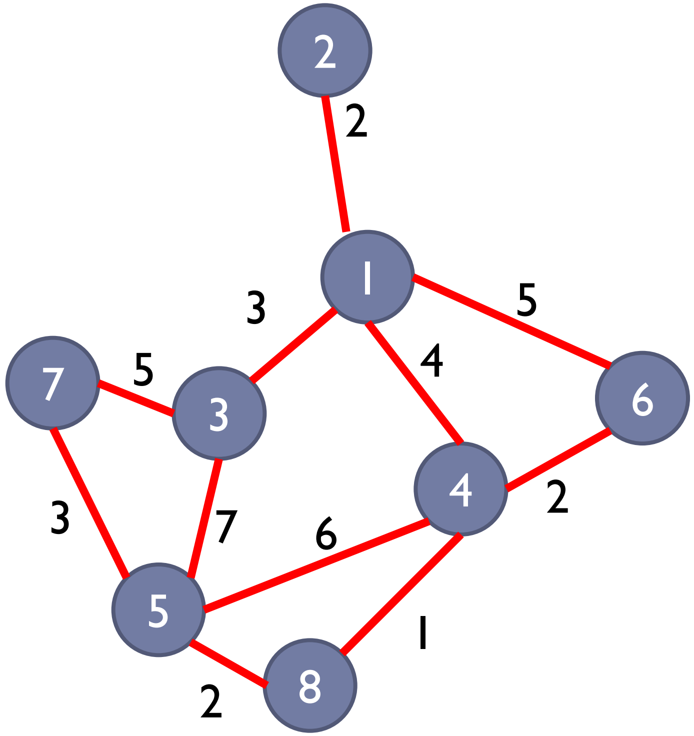
s – wierzchołek startowy

Wynik:

Najkrótsze drogi ze źródła s

do wszystkich pozostałych

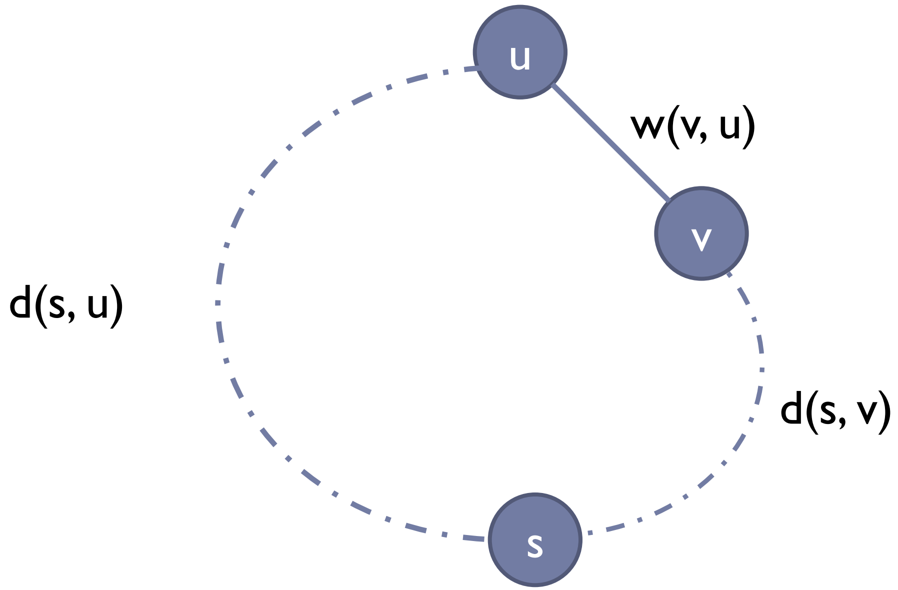
wierzchołków



Algorytm Dijkstry jest algorytmem zachłannym, który działa dla sieci o wagach nieujemnych.

Relaksacja - podstawa algorytmu

Przypuśćmy, że znaleźliśmy drogę z wierzchołka s do u i jej długość wynosi d(s,u). Jeśli znajdziemy drogę krótszą przez wierzchołek v poprzedzający u, to wybieramy tę krótszą drogę



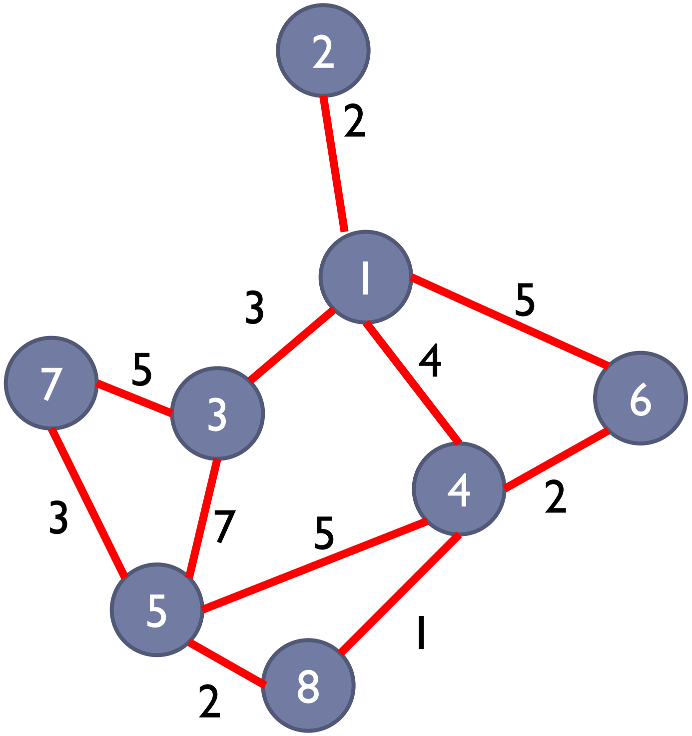
jeśli $d(s, u) \geq d(s, v) + w(v, u)$

to $d(s, u) = d(s, v) + w(v, u)$

Do u przyszliśmy z teraz v!

Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

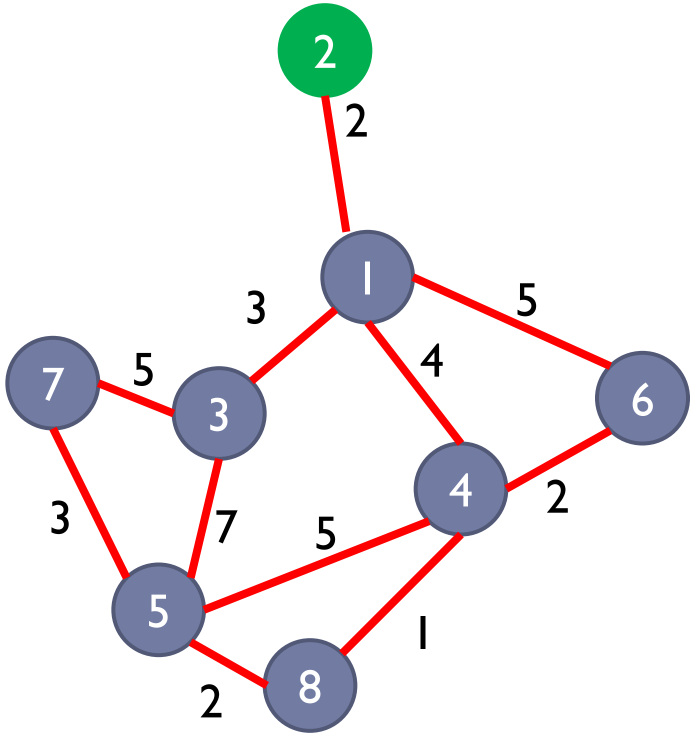


Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
∞	0	∞	∞	∞	∞	∞	∞

Odległość od źródła



Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0

Algorytm Dijkstry - zachłanny

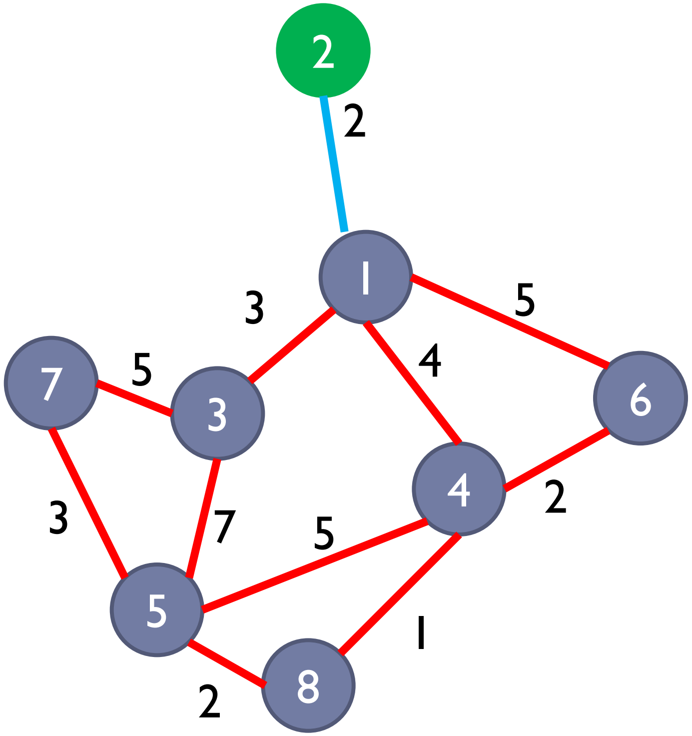
Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	∞	∞	∞	∞	∞	∞

Odległość od źródła

Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	0	0	0	0	0	0

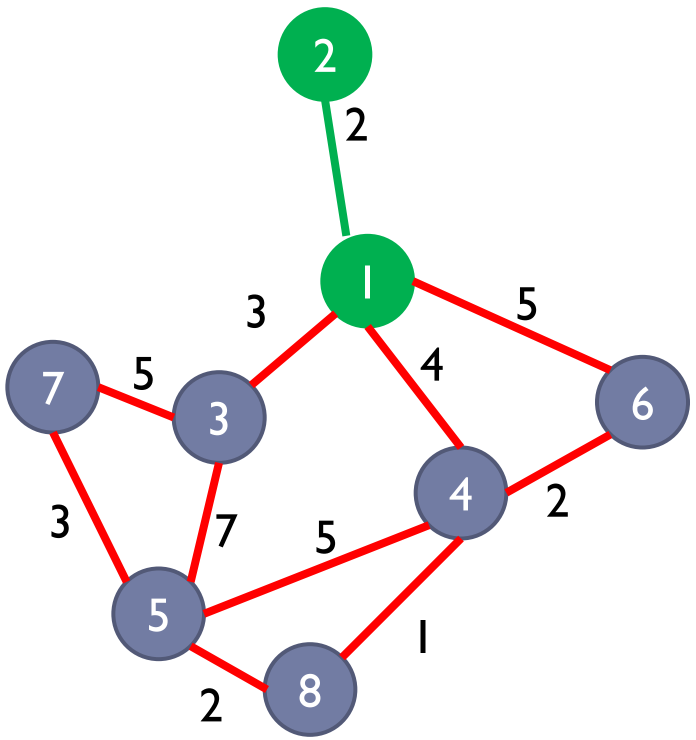


Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	∞	∞	∞	∞	∞	∞

Odległość od źródła



Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	0	0	0	0	0	0

Algorytm Dijkstry - zachłanny

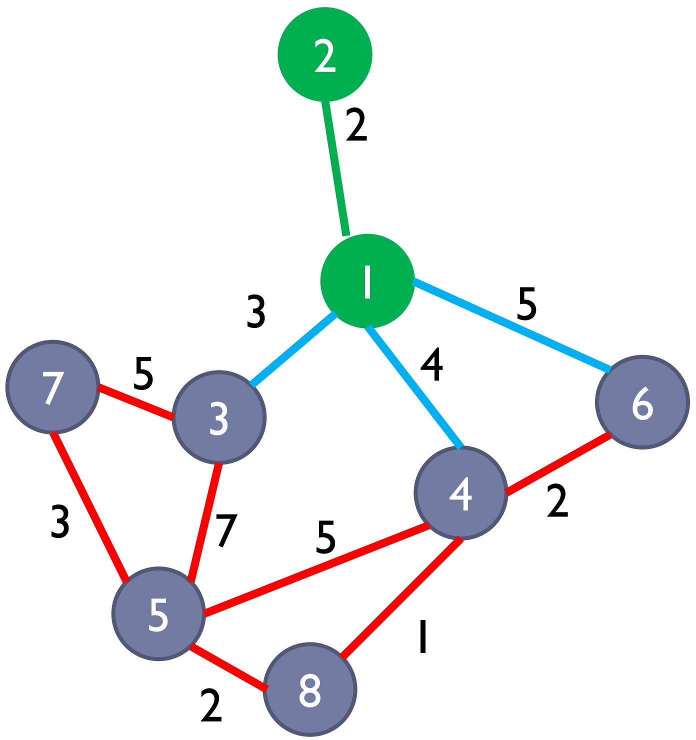
Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	∞	7	∞	∞

Odległość od źródła

Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	1	1	0	1	0	0

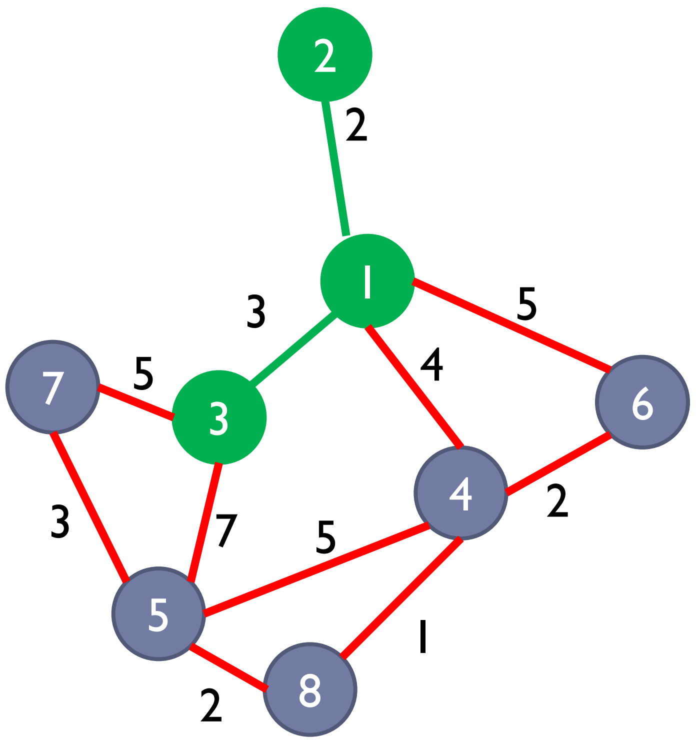


Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	∞	7	∞	∞

Odległość od źródła



Ojcowie - skąd przyszliśmy?

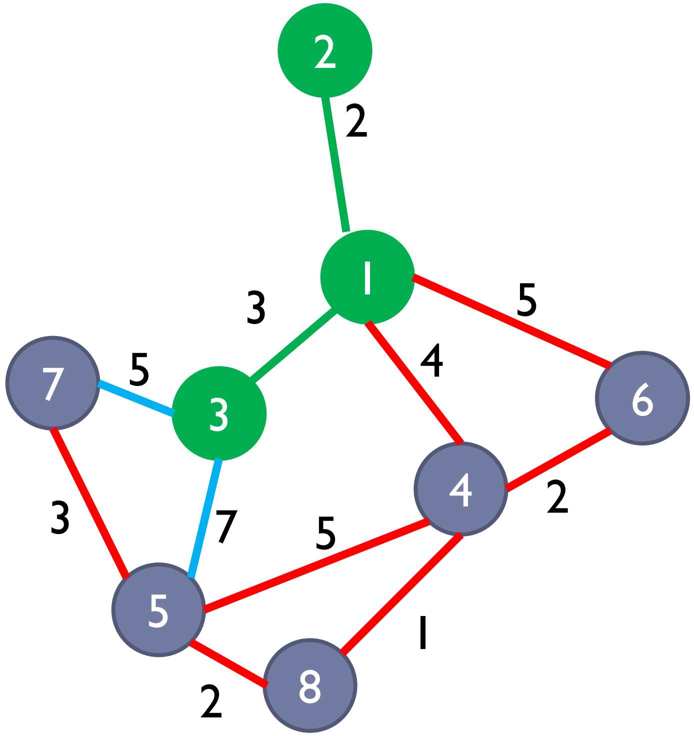
1	2	3	4	5	6	7	8
2	0	1	1	0	1	0	0

Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	12	7	10	∞

Odległość od źródła



Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	1	1	3	1	3	0

Algorytm Dijkstry - zachłanny

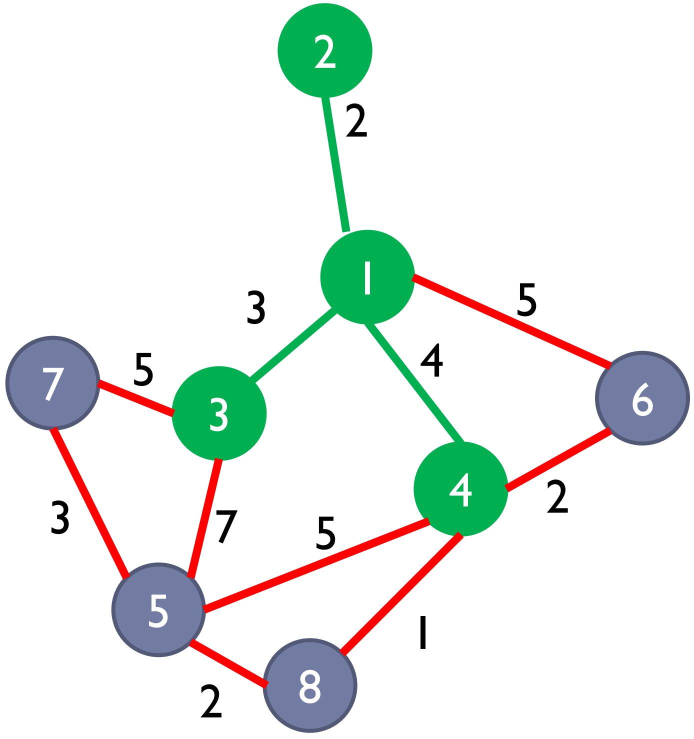
Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	**0**	**5**	**6**	12	7	10	∞

Odległość od źródła

Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	1	1	3	1	3	0



Algorytm Dijkstry - zachłanny

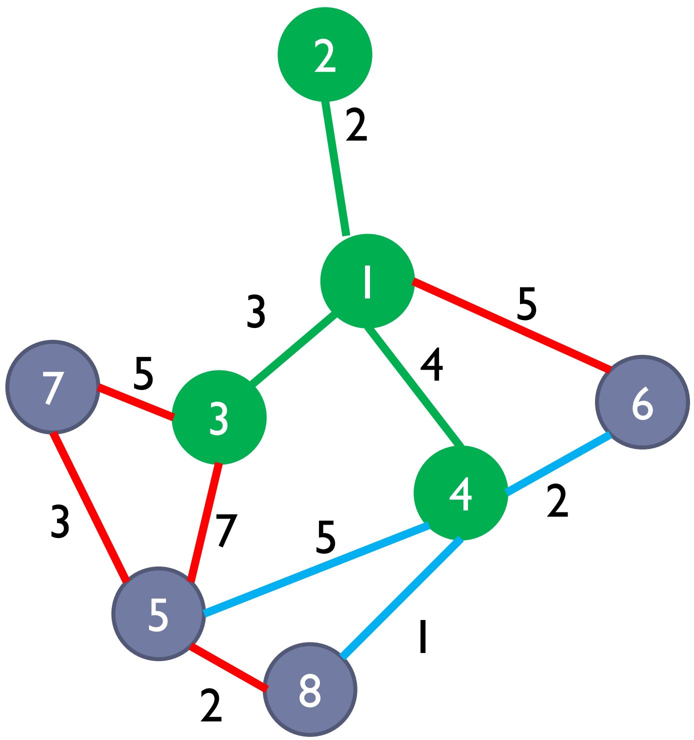
Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	11	7	10	7

Odległość od źródła

Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	1	1	4	1	3	0



Algorytm Dijkstry - zachłanny

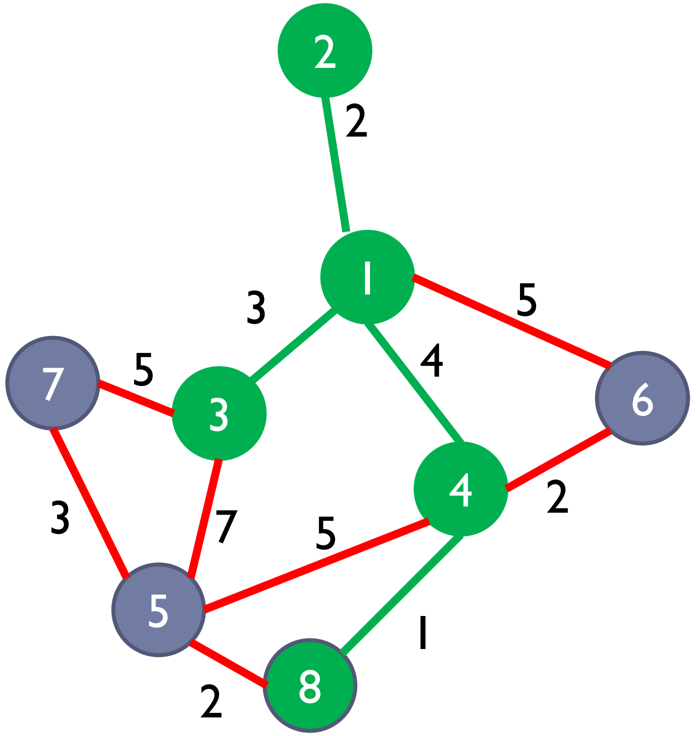
Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	11	7	10	7

Odległość od źródła

Ojcowie - skąd przyszliśmy?

1	2	3	4	5	6	7	8
2	0	1	1	4	1	3	4

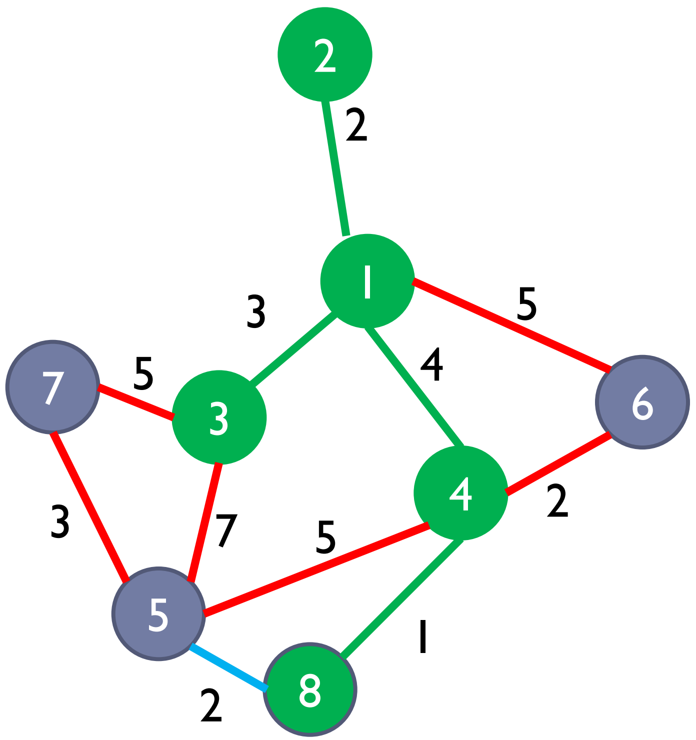


Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	9	7	10	7

Odległość od źródła



Ojcowie - skąd przyszliśmy?

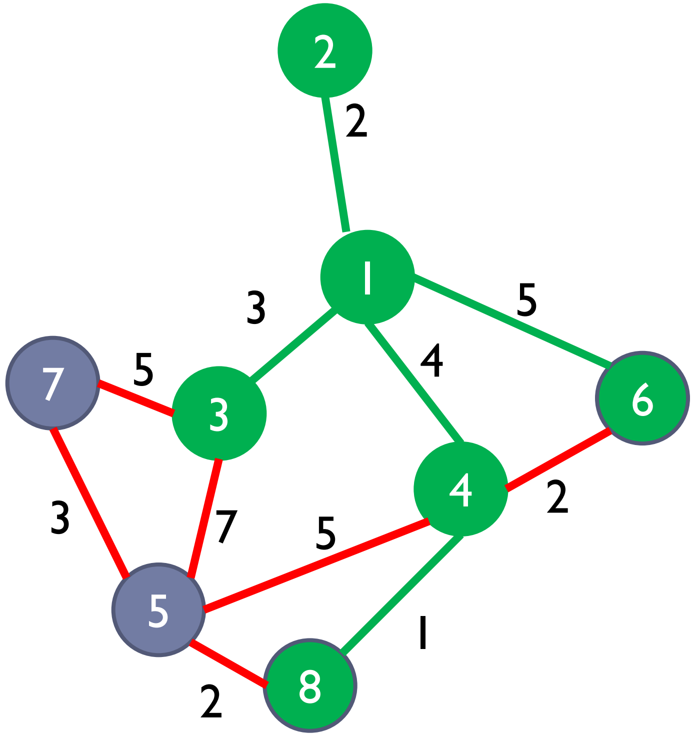
1	2	3	4	5	6	7	8
2	0	1	1	4	1	3	4

Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	**0**	**5**	**6**	9	**7**	10	**7**

Odległość od źródła



Ojcowie - skąd przyszliśmy?

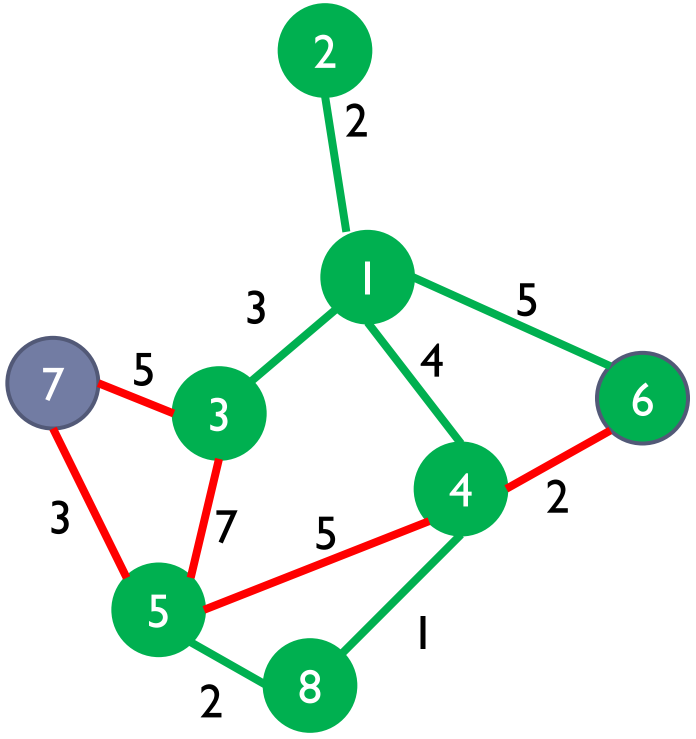
1	2	3	4	5	6	7	8
2	0	1	1	8	1	3	4

Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	9	7	10	7

Odległość od źródła



Ojcowie - skąd przyszliśmy?

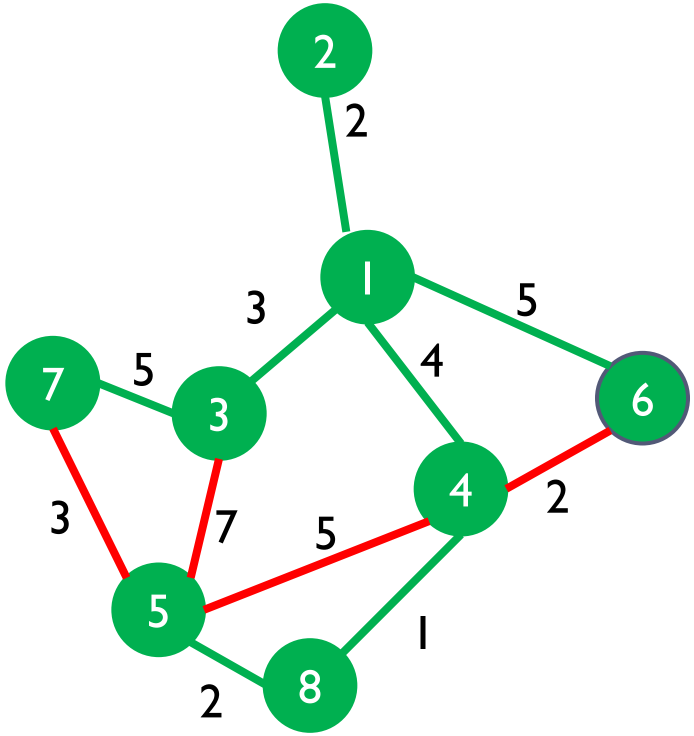
1	2	3	4	5	6	7	8
2	0	1	1	8	1	3	4

Algorytm Dijkstry - zachłanny

Jaka jest najkrótsza droga z 2 do 5?

1	2	3	4	5	6	7	8
2	0	5	6	9	7	10	7

Odległość od źródła



Ojcowie – drzewo najkrótszych dróg

1	2	3	4	5	6	7	8
2	0	1	1	8	1	3	4

Algorytm Dijkstry

Po zakończeniu działania algorytmu długość najkrótszej drogi z s do t jest można odczytać z tablicy/listy odległości, nazwijmy ją **d**.

Ciąg wierzchołków tworzących jedną z najkrótszych dróg można otrzymać z tablicy ojców, nazwijmy ją **p**, przechodząc tę drogę od tyłu, czyli z t do s.

Zatem najkrótsza droga z s do t to ciąg wierzchołków:

(s, p(p(...)), ..., p (p(t)), p(t), t).

Jeśli sieć nie zawiera drogi z s do t, to d(t) zachowuje wartość ∞.

Informacja o ustaleniu dla najkrótszej drogi dla wierzchołka zawarta jest w dodatkowej tablicy/liście, nazwijmy ją **f**.

Wierzchołek, któremu ostatnio ustalono najkrótszą odległość (cechę) oznaczamy przez **r**.

Ćwiczenie: Sprawdź, że algorytm działa źle dla grafu z ujemnymi wagami.

Algorytm Dijkstry - pseudokod

Inicjalizacja:

```
for v∈V do
begin
    d(v) ← ∞;
    f(v) ← false;
    p(v) ← -1;
end;
d(s) ← 0;
f(s) ← true;
r ← s;
```

Iteracja:

```
while f(t) = false do
begin
    for v∈N⁺(r) if not f(v) do
    begin
        newlabel ← d(r)+w[r,v];
        if newlabel< d(v) then
        begin
            d(v) ← newlabel;
            p(v) ← r;
        end;
    end;
    znaleźć y – wierzchołek o najmniejszej
    wartości w d różnej od ∞;
    f(y) ← true;
    r ← y;
end;
```

Algorytm Dijkstry – analiza algorytmu

Pamięć zajmowana przez dane to n^2+3n, $2m+n^2+3n$ lub $2*2m+3n$ w zależności od reprezentacji.

- Czas wykonania inicjalizacji jest proporcjonalny do n.
- Zewnętrzna pętla kroku iteracyjnego pesymistycznie wykonuje się n–1 razy.
- Podczas każdego wykonania zewnętrznej pętli należy przeglądnąć jeden wiersz macierzy wag W odpowiadający wierzchołkowi r i uaktualnić tablice d i p – czas proporcjonalny do n.
- Aby znaleźć najmniejszą cechę w i-tej iteracji trzeba wykonać n–i–1 porównań.

Całkowity czas wykonania algorytmu Dijkstry **z szukaniem liniowym minimum** jest równy $O(n^2)$.

- Jeśli sieć jest pełna lub jest dana w postaci macierzy wag, to algorytm o złożoności $O(n^2)$ jest najlepszym, jakiego można oczekiwać.
- Jeśli sieć jest rzadka to można zmniejszyć czas obliczeń wybierając inną reprezentację np. dwie tablice lub listy sąsiadów.
- Klasyczny algorytm Dijkstry znajduje pojedynczą najkrótszą drogę z s do t. Jeśli będziemy kontynuować obliczenia, aż każdy wierzchołek otrzyma stałą cechę, to otrzymamy algorytm wyznaczający najkrótsze drogi ze źródła s do wszystkich pozostałych wierzchołków w sieci. Wymaga to również $O(n^2)$ operacji.

Algorytm Dijkstry - złożoność

Dane:

Graf G=(V,E) może być skierowany lub nie, o nieujemnych wagach łuków (lista sąsiadów) oraz wyróżniony wierzchołek s nazywany źródłem.

Wynik:

Najkrótsza droga z wierzchołka s do pozostałych wierzchołków.

Złożoność obliczeniowa

- jeśli za każdym razem będziemy naiwnie liniowo wyszukiwać najmniejszą wartość w liście to algorytm jest rzędu $O(n^2)$,
- jeśli będziemy pobierać minimum z kopca, otrzymujemy złożoność O(E logV),
- jeśli zastosujemy kopiec Fibonacciego, mamy O(E+VlogV).

Algorytm Moore'a-Bellmana 1957/58

Znajdowanie najkrótszej drogi z ustalonego wierzchołka początkowego s (źródła) do innego ustalonego wierzchołka t (odpływu) w sieci skierowanej o **dowolnych** wagach łuków.

Aby prawidłowo obliczyć najkrótszą drogę od wierzchołka s do t, jeśli wagi krawędzi są dowolne, również ujemne, nie możemy ustalać żadnej z cech przed zakończeniem działania algorytmu. Metoda ta zwana jest metodą poprawiania cech.

Inicjalizacja algorytmu przebiega podobnie, jak w algorytmie Dijkstry.

W kroku iteracyjnym cecha d(v) jest uaktualniana tak, aby jej wartość była równa bieżącej odległości v od s, a p(v) staje się numerem wierzchołka bezpośrednio poprzedzającego v na bieżącej, najkrótszej drodze z s do v.

Iteracja ta może być przedstawiona zwięźle:

while istnieje łuk (u,v) spełniający $d(u) + w[u,v] < d(v)$ do

begin

  $d(v) \leftarrow d(u) + w[u,v]$;

  $p(v) \leftarrow u$;

end;

Algorytm Moore'a-Bellmana - struktury danych

Dane: Sieć skierowana G (V, E) o dowolnych wagach łuków w postaci macierzy wag W oraz wyróżnione wierzchołki s i t.

Wynik: Najkrótsza droga z ustalonego wierzchołka s do innego ustalonego wierzchołka t.

d tablica/lista, w której pamiętamy wartości cech wierzchołków.

p tablica/lista bezpośrednich poprzedników każdego wierzchołka na najkrótszej drodze z s do t.

kolejka Q przechowuje wierzchołki, które mają być badane.

Kolejka, początkowo zawiera tylko wierzchołek s.

Wierzchołek v wchodzi do kolejki wtedy, gdy wartość d(v) ulega zmniejszeniu i v nie należy do Q.

Jeden wierzchołek może być dołączany do kolejki i usuwany z niej wiele razy, za każdym razem, gdy została znaleziona krótsza droga – działanie nieefektywne, zostało później poprawione.

Poprawka d'Esopo i Papiego 1980

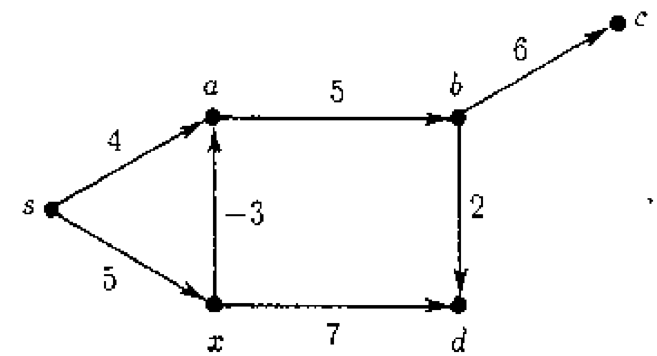
Wierzchołek v umieszczamy:

- na końcu kolejki, jeśli v nie został wcześniej osiągnięty,
- na początku kolejki, jeśli już do niej należał i był badany.

W drugim przypadku wierzchołek v będzie szybko przebadany ponownie i zostaną zmniejszone cechy wszystkim tym wierzchołkom, które są osiągalne za jego pośrednictwem.

Dzięki poprawce ulega zmniejszeniu liczba powrotów wierzchołka v do kolejki.

Przykład działania algorytmu Moore'a-Bellmana



Q	dist s	a	b	c	d	x
s	0	∞	∞	∞	∞	∞
a, x	0	4	∞	∞	∞	5
x, b	0	4	9	∞	∞	5
a, b, d	0	2	9	∞	12	5
b, d	0	2	7	∞	12	5
d, c	0	2	7	13	9	5
c	0	2	7	13	9	5
	0	2	7	13	9	5

Algorytm Moore'a-Bellmana - pseudokod

Inicjalizacja:

```
for v∈V do
begin
    d(v) ← ∞;
    p(v) ← -1;
end;
d(s) ← 0;
Enqueue(Q,s);
(do kolejki)
```

Iteracja:

```
while Q ≠ ∅ do
begin
    for każdego łuku (u,v) do
    begin
        newlabel ← d(u)+w[u,v];
        if newlabel< d(v) then
        begin
            d(v) ← newlabel;
            p(v) ← u;
            if v nie należał dotychczas do Q then Enqueue[Q,v]
            else
            if v był już w Q, ale teraz v nie należy do Q then
            dołącz v na początku Q;
        end;
    end;
end;
```

Algorytm Moore'a-Bellmana – złożoność

Złożoność algorytmu trudno określić jednoznacznie:

- Jeśli sieć jest mała i gęsta to szybszy jest algorytm Dijkstry, o ile wagi nie są ujemne.
- Jeśli sieć ma małą liczbę łuków w porównaniu z liczbą wierzchołków, to algorytm Moore'a-Bellmana jest szybszy od prawie każdego innego algorytmu znajdowania najkrótszych dróg z ustalonego wierzchołka do wszystkich pozostałych wierzchołków w sieci.
- Jeśli każdy wierzchołek trafi do kolejki dokładnie raz, to złożoność czasowa będzie proporcjonalna do liczby łuków w sieci.
- Jeśli każdy wierzchołek trafia do kolejki maksymalna liczbę razy, czas obliczeń rośnie wykładniczo wraz ze wzrostem liczby wierzchołków.

Algorytm wyznacza długość najkrótszej drogi nie tylko z s do t, ale do wszystkich pozostałych wierzchołków w sieci. Wynikiem działania algorytmu jest więc **drzewo najkrótszych dróg** z korzeniem w s.

Algorytm Floyda–Warshalla 1962

Wyznaczanie najkrótszych dróg **między każda parą wierzchołków** w sieci skierowanej o dowolnych wagach łuków, nie zwierającej cykli ujemnej długości.

Sieć skierowana o n wierzchołkach zawiera n(n-1) takich dróg, a w sieci bez skierowania mamy n(n-2)/2 dróg.

Do szukania najkrótszych dróg między każdą parą wierzchołków można zastosować dla każdego wierzchołka poznane do tej pory algorytmy szukające:

- najkrótszych dróg – algorytm Dijkstry nie działa jednak na grafach z krawędziami ujemnymi,
- algorytm Moore'a – Bellmana nie zawsze jest efektywny w zależności od rozmiarów danych.

Algorytm Floyda – Warshalla dopuszcza ujemne wagi łuków.

Podstawowym krokiem algorytmu jest włączanie do drogi jednego lub więcej wierzchołków, jeśli tylko jest to korzystne.

Algorytm Floyda–Warshalla – opis działania

Niech G(V,E) będzie siecią skierowaną o n wierzchołkach z funkcją wag w.

Graf ten jest reprezentowany przez macierz wag W[i,j] o rozmiarze n x n.

Konstruujemy ciąg n macierzy $\mathbf{W^{(1)}}$, $\mathbf{W^{(2)}}$, **… ,** $\mathbf{W^{(n)}}$.

Element $W[i,j]^{(k)}$ w macierzy $W^{(k)}$ jest długością najkrótszej drogi spośród wszystkich dróg z i do j, których wierzchołki pośrednie należą do zbioru 1, 2, …, k.

Macierz $W^{(k)}$ może być więc utworzona z macierzy $W^{(k-1)}$ w następujący sposób:

$W[i,j]^{(0)} = w[i,j]$,

$W[i,j]^{(k)} = \min \{W[i,j]^{(k-1)}, W[i,k]^{(k-1)} + W[k,j]^{(k-1)}\}$, dla k = 1,2,… , n.

W pierwszej iteracji, do drogi z i do j jest włączany wierzchołek 1, jeśli W[i,j] > W[i,1] +W[1,j], w drugiej może jest włączany drugi wierzchołek, itd.

Zakładamy, że waga łuku nieistniejącego jest równa ∞, x+ ∞ = ∞, min{x, ∞} = x, dla wszystkich liczb rzeczywistych x.

Wszystkie macierze $W^{(k)}$ pamiętane są kolejno w miejscu jednej macierzy W.

Algorytm Floyda–Warshalla - pseudokod

```
for  k←1  to  n  do
        for  i←1  to  n  do
                if  W[i,k] ≠ ∞  then
                        for  j←1  to  n  do  W[i,j]← min{ W[i,j], W[i,k] + W[k,j] }
```

Element $W[i,i]^{(n)}$ jest długością najkrótszego cyklu przechodzącego przez wierzchołek i, a elementy $W[i,j]^{(n)}$ poza przekątną są długościami najkrótszych dróg między wierzchołkami.

Najkrótsze drogi mogą być otrzymane z macierzy dróg P, zwanej macierzą optymalnych decyzji, w której P[i,j] jest przedostatnim wierzchołkiem na najkrótszej drodze z i do j. Aby wyznaczyć tę macierz należy na początku ustalić:

$$P[i,j] \leftarrow i, \quad \text{jeśli } W[i,j] \neq \infty,$$

$$P[i,j] \leftarrow 0, \quad \text{jeśli } W[i,j] = \infty.$$

W k-tej iteracji, jeśli wierzchołek k zostaje włączony do drogi z i do j, tzn., gdy

$W[i, j] > W[i, k] + W[k, j]$, przyjmujemy $P[i,j] \leftarrow P[k,j]$.

Wierzchołki najkrótszej drogi mogą być otrzymane z tej tablicy:

$$v_q=P[i, j],\ v_{q-1}=P[i, v_q],\ v_{q-2}=P[i, v_{q-1}],\ \ldots,\ i = P[i, v_1]$$

Algorytm Floyda–Warshalla – złożoność i efektywność

Potrzebujemy n^2 słów pamięci na zapamiętanie jednej macierzy stopnia n.

Wszystkie macierze pośrednie i końcowa pamiętamy w miejscu danej macierzy W. Dodatkowo potrzeba n^2 słów pamięci dla macierzy P.

Czas działania algorytmu wynosi **O(n^3),** bez względu na gęstość rozpatrywanej sieci.

Porównanie czasu obliczeń (efektywności) algorytmów najkrótszych dróg w grafach pełnych:

Liczba wierzchołków	Procedura		
	DIJKSTRA	PDM	FLOYD
40	0.527	0.708	0.646
60	1.767	2.489	2.156
80	4.208	6.038	5.078
100	8.052	11.601	9.862

Dziękuję z uwagę

dr Anna Beata Kwiatkowska

aba@mat.umk.pl

tel. 602 184 813