Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
AGH University of Science
and Technology

Grafy i ich zastosowania

Zestaw 4

Elzbieta.Strzalka@fis.agh.edu.pl
p. 232/D-10

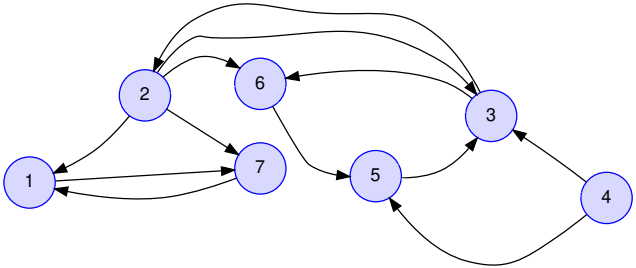
AGH

Zestaw 4, zadanie 1

Napisać program do kodowania grafów skierowanych (digrafów) i do generowania losowych digrafów z zespołu $G(n, p)$.

Lista sąsiedztwa

- Graf **skierowany** $\equiv$ digraf (***di**rected **graph***).
- Krawędź skierowana $\equiv$ **łuk**.
- **Digraf prosty** – wszystkie łuki są różne, brak pętli.

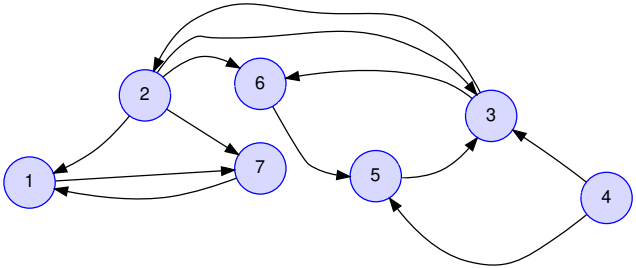


Lista o n wierszach

v	**Sąsiedzi** v
1	
2	
3	
4	
5	
6	
7	

Lista sąsiedztwa

- Graf **skierowany** $\equiv$ digraf (***di**rected **graph***).
- Krawędź skierowana $\equiv$ **łuk**.
- **Digraf prosty** – wszystkie łuki są różne, brak pętli.

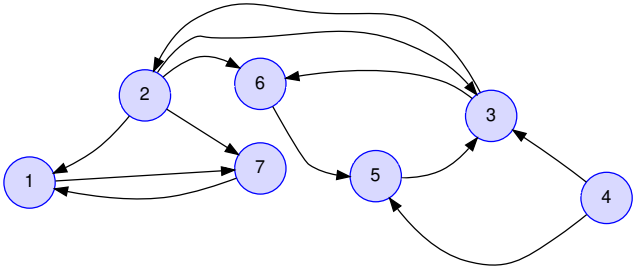


Lista o n wierszach

v	**Sąsiedzi** v
1	7
2	
3	
4	
5	
6	
7	

Lista sąsiedztwa

- Graf **skierowany** $\equiv$ digraf (***di**rected **graph***).
- Krawędź skierowana $\equiv$ **łuk**.
- **Digraf prosty** – wszystkie łuki są różne, brak pętli.

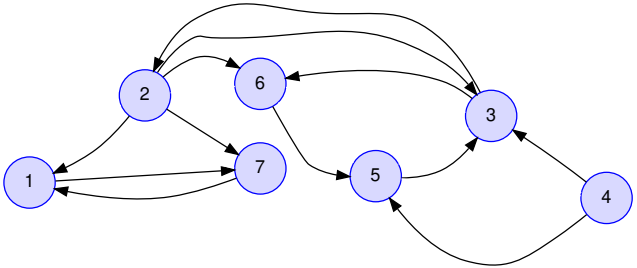


Lista o n wierszach

v	**Sąsiedzi** v
1	7
2	1, 3, 6, 7
3	
4	
5	
6	
7	

Lista sąsiedztwa

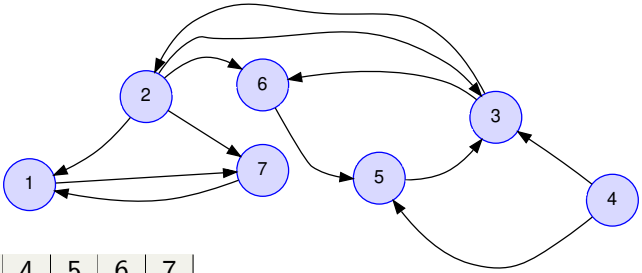
- Graf **skierowany** $\equiv$ digraf (***di**rected **graph***).
- Krawędź skierowana $\equiv$ **łuk**.
- **Digraf prosty** – wszystkie łuki są różne, brak pętli.



Lista o n wierszach

v	**Sąsiedzi** v
1	7
2	1, 3, 6, 7
3	2, 6
4	3, 5
5	3
6	5
7	1

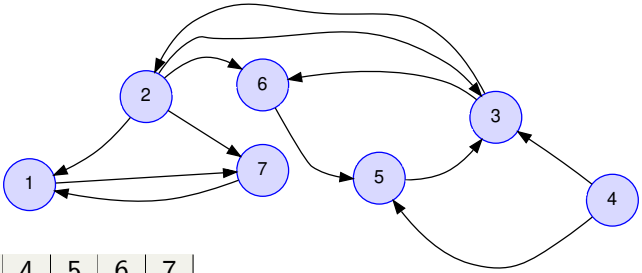
Macierz sąsiedztwa



	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

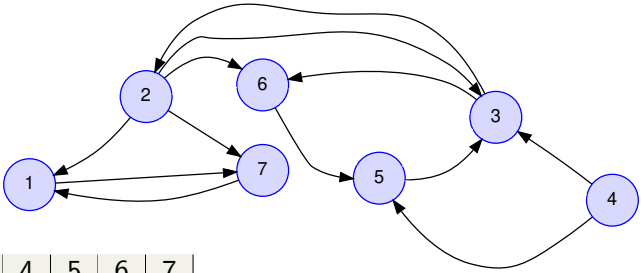
Macierz sąsiedztwa



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

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

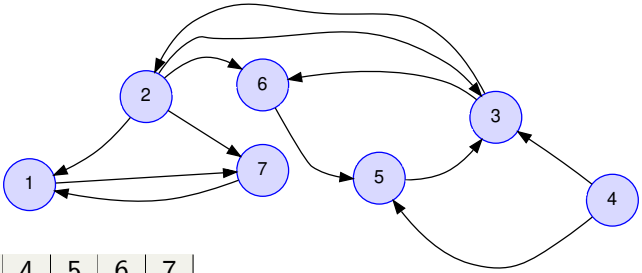
Macierz sąsiedztwa



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

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

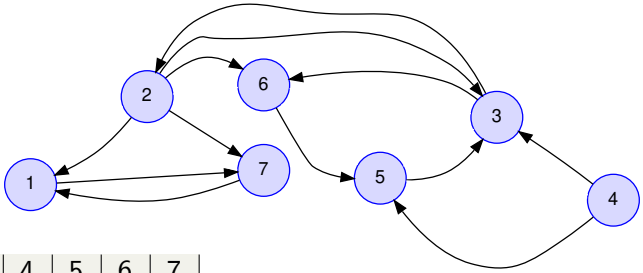
Macierz sąsiedztwa



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

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

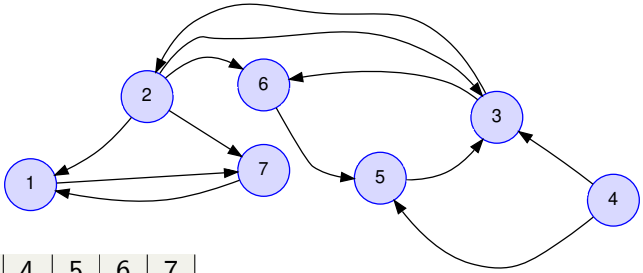
Macierz sąsiedztwa



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

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

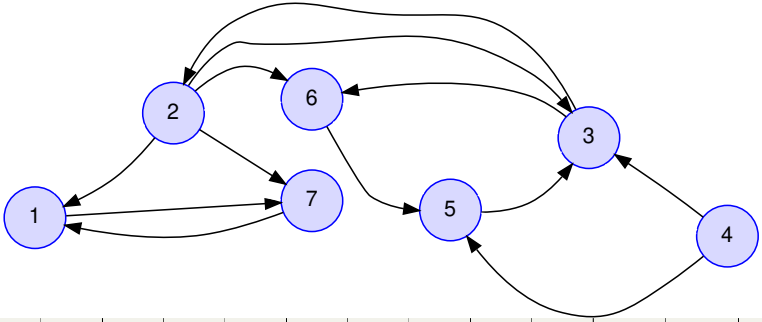
Macierz sąsiedztwa



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

- Macierz kwadratowa $n \times n$
- **Brak symetrii**: możliwe jednostronne sąsiedztwo.

Macierz incydencji



	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}	L_{11}	L_{12}
1	**-1**	**1**	0	0	0	0	0	0	0	0	0	**1**
2	0	**-1**	**-1**	**-1**	**-1**	**1**	0	0	0	0	0	0
3	0	0	**1**	0	0	**-1**	**-1**	**1**	0	**1**	0	0
4	0	0	0	0	0	0	0	**-1**	**-1**	0	0	0
5	0	0	0	0	0	0	0	0	**1**	**-1**	**1**	0
6	0	0	0	**1**	0	0	**1**	0	0	0	**-1**	0
7	**1**	0	0	0	**1**	0	0	0	0	0	0	**-1**

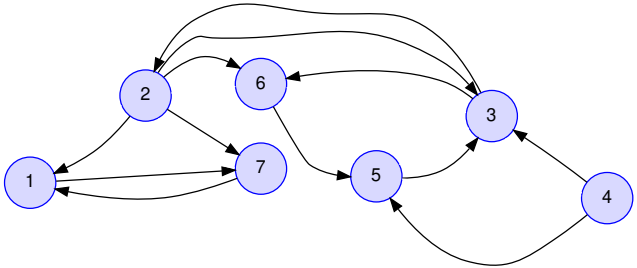
- Macierz $n \times k$
- Kodowanie **zwrotu**: -1 przy źródle.
- Kolejność kolumn **dowolna**.
- $G(n, p)$: p jest prawdopodobieństwem istnienia każdego z łuków.

Zestaw 4, zadanie 2

Zaimplementować algorytm Kosaraju do szukania silnie spójnych składowych na digrafie i zastosować go do digrafu losowego.

Silnie spójne składowe

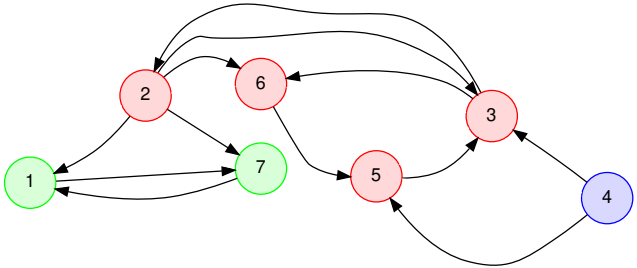
- **Graf skierowany jest silnie spójny** $\Leftrightarrow$ między każdą parą wierzchołków istnieje ścieżka (czyli: istnieje ścieżka $u \to v$, ale też $v \to u$).
- **Niespójny** silnie digraf składa się ze **składowych**, które są **silnie spójne**.
- Jeżeli dwa wierzchołki należą do tej samej silnie spójnej składowej, to żadna ścieżka między nimi nie zawiera wierzchołka spoza tej składowej.



Czy silnie spójny?

Silnie spójne składowe

- **Graf skierowany jest silnie spójny** $\Leftrightarrow$ między każdą parą wierzchołków istnieje ścieżka (czyli: istnieje ścieżka $u \to v$, ale też $v \to u$).
- **Niespójny** silnie digraf składa się ze **składowych**, które są **silnie spójne**.
- Jeżeli dwa wierzchołki należą do tej samej silnie spójnej składowej, to żadna ścieżka między nimi nie zawiera wierzchołka spoza tej składowej.

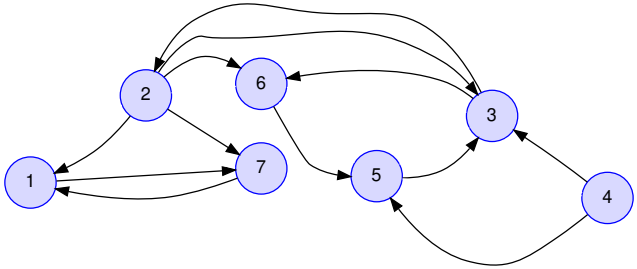


Czy silnie spójny?
Nie.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju

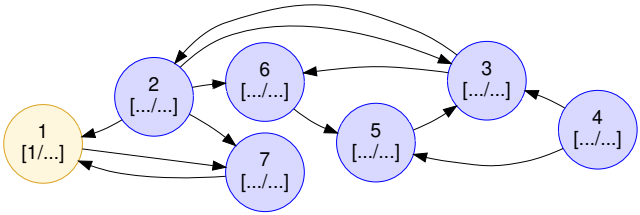
- Podstawa: silnie spójna składowa zawiera te wierzchołki, które są nawzajem osiągalne w grafie G oraz w grafie transponowanym G^{T}.
- **Dwa przeszukiwania w głąb**:
  1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f.
  2. **Transpozycja grafu**: G^{T} (te same krawędzie, przeciwne zwroty).
  3. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej.



Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



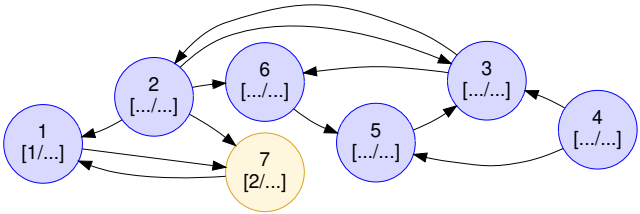
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



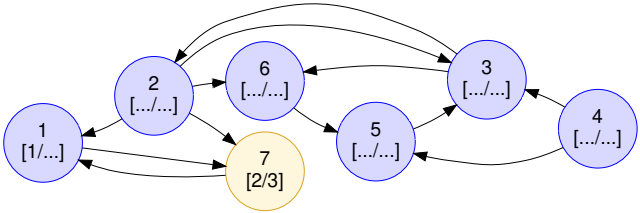
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



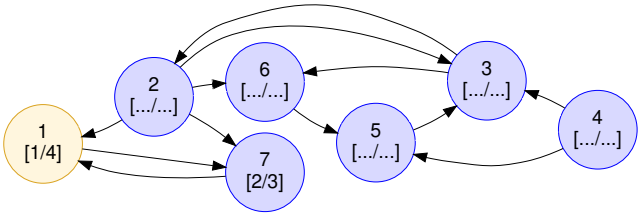
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



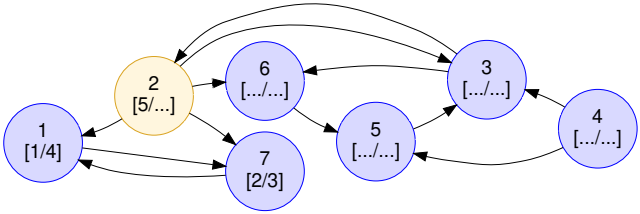
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



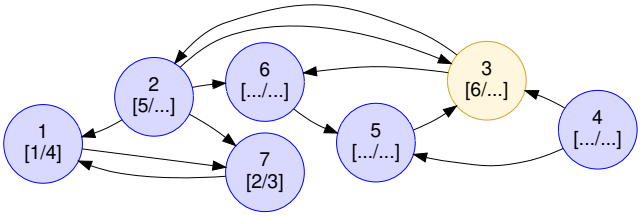
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



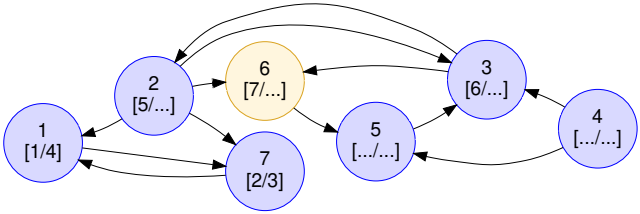
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



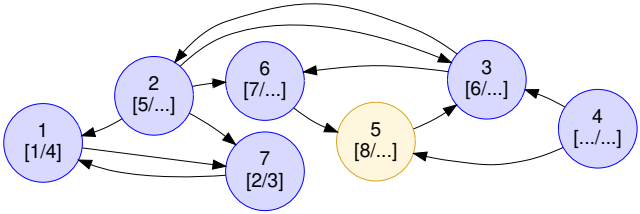
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



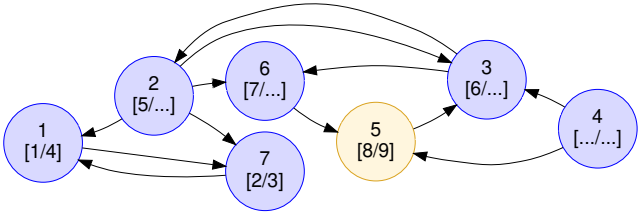
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



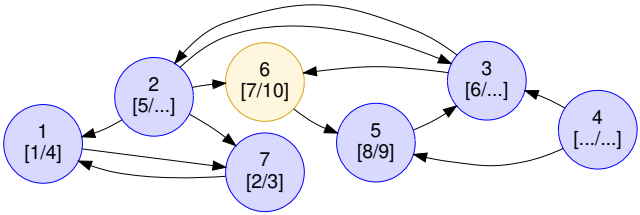
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



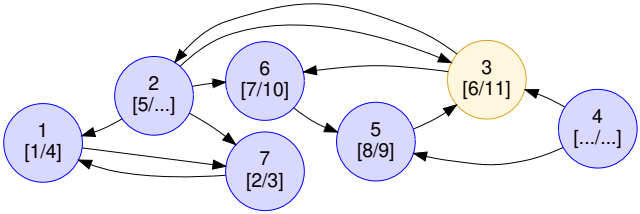
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



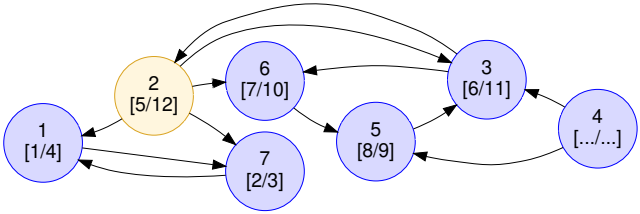
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



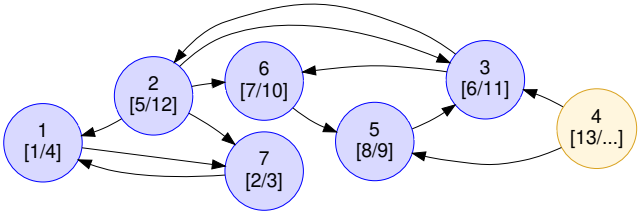
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



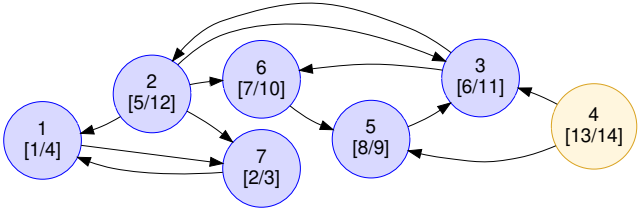
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



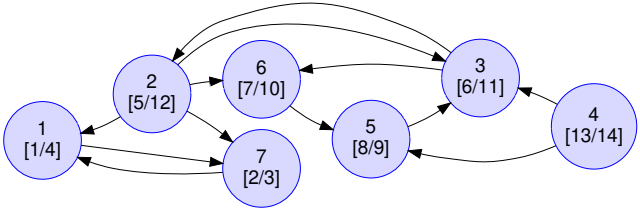
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 1. etap

1. **Pierwsze DFS**: oznaczenie czasów odwiedzenia d i przetworzenia f ($O(n+k)$).



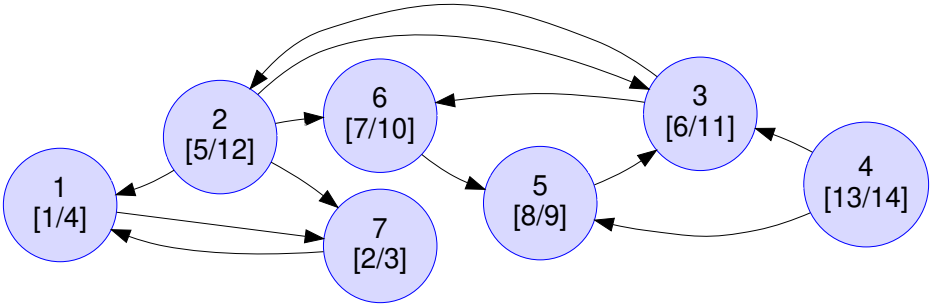
Zapis do tablic

- Start: dowolny wierzchołek.
- $d[u]$ zapisujemy w momencie odwiedzenia u.
- $f[u]$ zapisujemy w momencie zakończenia przetwarzania u.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 2. etap

1. **Transpozycja grafu**: G^{T} (te same krawędzie, przeciwne zwroty): $O(n+k)$.

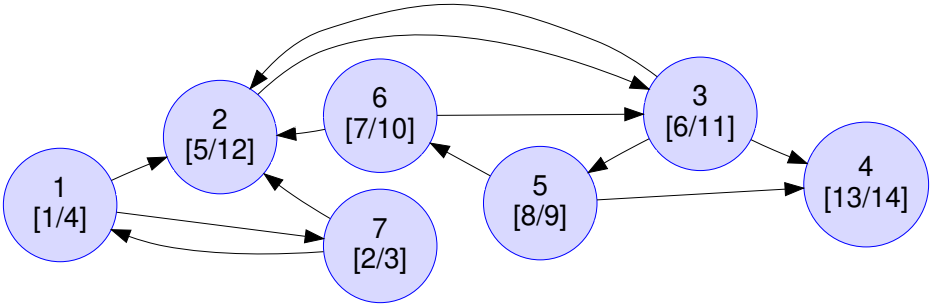


Graf G

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 2. etap

1. **Transpozycja grafu**: G^{T} (te same krawędzie, przeciwne zwroty): $O(n+k)$.

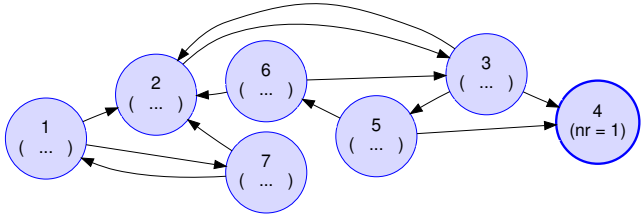


Graf G^{T}

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

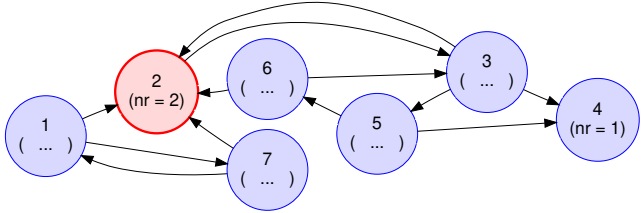
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, **2**, 3, 6, 5, 1, 7.

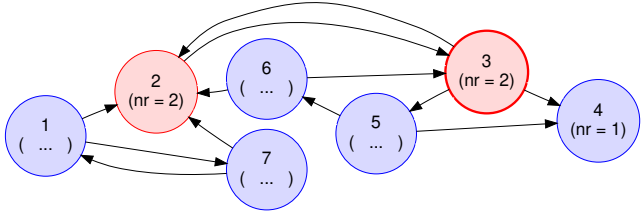
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

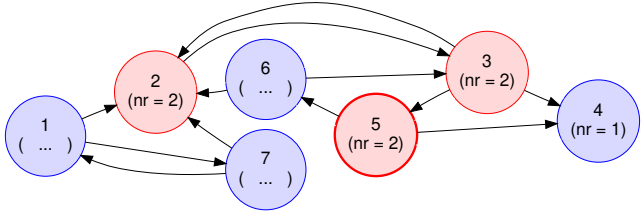
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

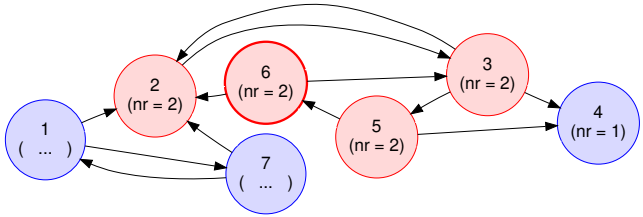
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

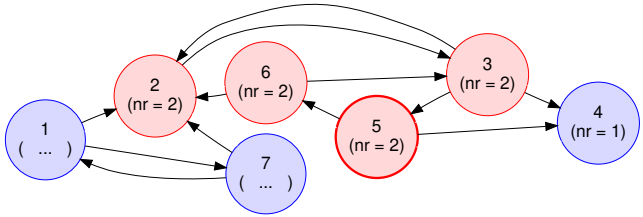
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

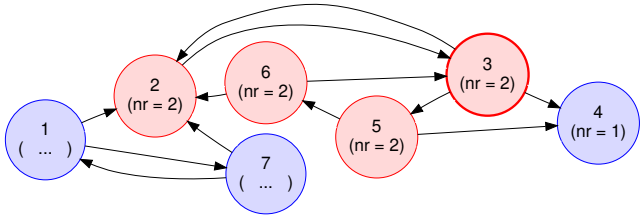
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

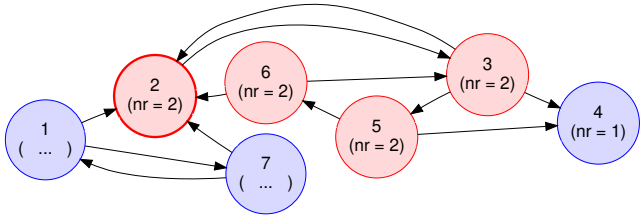
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

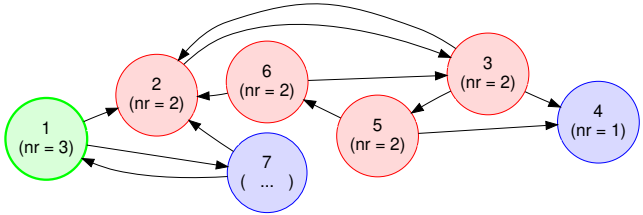
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, **1**, 7.

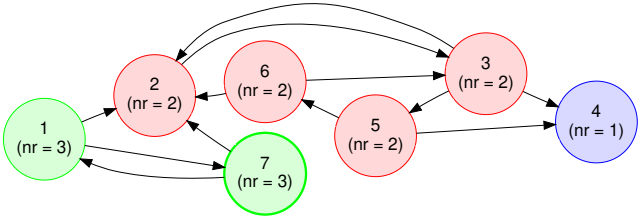
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.
- Składowa 3, wierzchołki: 1

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

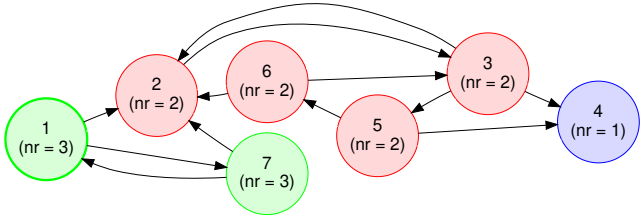
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.
- Składowa 3, wierzchołki: 1, 7.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

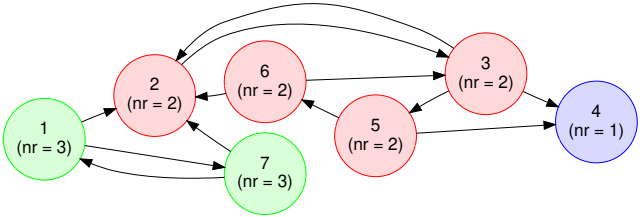
Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.
- Składowa 3, wierzchołki: 1, 7.

Algorytm oznaczania silnie spójnych składowych: algorytm Kosaraju

Algorytm Kosaraju: 3. etap

1. **Drugie DFS**: w kolejności malejących czasów f, oznaczenie numeru składowej ($O(n+k)$).



Kolejność przeszukiwania

4, 2, 3, 6, 5, 1, 7.

Znalezione spójne składowe

- Składowa 1, wierzchołki: 4.
- Składowa 2, wierzchołki: 2, 3, 5, 6.
- Składowa 3, wierzchołki: 1, 7.

AGH

Zestaw 4, zadanie 3

Wykorzystując algorytmy z powyższych punktów, wygenerować losowy silnie spójny digraf. Łukom tego digrafu przypisać losowe wagi będące liczbami całkowitymi z zakresu $[-5, 10]$. Zaimplementować algorytm Bellmana-Forda do znajdowania najkrótszych ścieżek od danego wierzchołka.

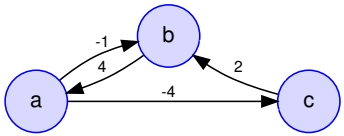
Losowy silnie spójny digraf z ujemnymi wagami

Jak znaleźć najkrótsze ścieżki z zadanego wierzchołka?

- Algorytm Dijkstry?

Algorithm: `dijkstra`(G, w, s)

```
1: init(G, s)
2: S ← ∅
3: while S ≠ zbiór wszystkich wierzchołków G do
4:     u ← wierzchołek o najmniejszym d_s[u] z niegotowych (u ∉ S)
5:     S ← S ∪ u
6:     for każdy wierzchołek v ∉ S będący sąsiadem u do
7:         relax(u, v, w)
8:     end for
9: end while
```



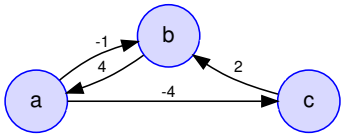
Losowy silnie spójny digraf z ujemnymi wagami

Jak znaleźć najkrótsze ścieżki z zadanego wierzchołka?

- ~~Algorytm Dijkstry~~ – działa tylko przy nieujemnych wagach oraz nie wykrywa ujemnego cyklu (wykonuje k relaksacji).

Algorithm: `dijkstra`(G, w, s)

```
1: init(G, s)
2: S ← ∅
3: while S ≠ zbiór wszystkich wierzchołków G do
4:     u ← wierzchołek o najmniejszym d_s[u] z niegotowych (u ∉ S)
5:     S ← S ∪ u
6:     for każdy wierzchołek v ∉ S będący sąsiadem u do
7:         relax(u, v, w)
8:     end for
9: end while
```



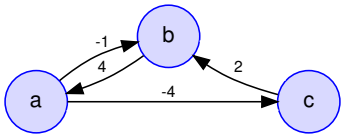
Losowy silnie spójny digraf z ujemnymi wagami

Jak znaleźć najkrótsze ścieżki z zadanego wierzchołka?

- **Algorytm Bellmana-Forda** – działa również przy ujemnych wagach, wykrywa ujemny cykl. Ale: $(n-1)+1$ relaksacji każdej z k krawędzi $\Rightarrow O(n \cdot k)$.

Algorithm: `bellman_ford`(G, w, s)

```
1:  init(G, s)
2:  for i ← 1 to n − 1 do
3:      for każda krawędź (u, v) należąca do grafu G do
4:          relax(u, v, w)
5:      end for
6:  end for
7:  for każda krawędź (u, v) należąca do grafu G do
8:      if d_s[v] > d_s[u] + w[u][v] then
9:          return FALSE
10:     end if
11: end for
12: return TRUE
```



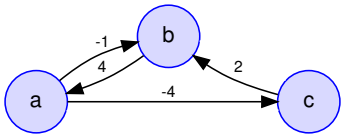
Losowy silnie spójny digraf z ujemnymi wagami

Jak znaleźć najkrótsze ścieżki z zadanego wierzchołka?

- **Algorytm Bellmana-Forda** – działa również przy ujemnych wagach, wykrywa ujemny cykl. Ale: $(n-1)+1$ relaksacji każdej z k krawędzi $\Rightarrow O(n \cdot k)$.

Algorithm: `bellman_ford`(G, w, s)

```
1:  init(G, s)
2:  for i ← 1 to n − 1 do
3:      for każda krawędź (u, v) należąca do grafu G do
4:          relax(u, v, w)
5:      end for
6:  end for
7:  for każda krawędź (u, v) należąca do grafu G do
8:      if d_s[v] > d_s[u] + w[u][v] then
9:          return FALSE
10:     end if
11: end for
12: return TRUE
```



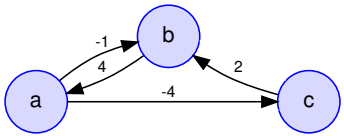
Losowy silnie spójny digraf z ujemnymi wagami

Jak znaleźć najkrótsze ścieżki z zadanego wierzchołka?

- **Algorytm Bellmana-Forda** – działa również przy ujemnych wagach, wykrywa ujemny cykl. Ale: $(n-1)+1$ relaksacji każdej z k krawędzi $\Rightarrow O(n \cdot k)$.

Algorithm: `bellman_ford`(G, w, s)

```
1:  init(G, s)
2:  for i ← 1 to n − 1 do
3:      for każda krawędź (u, v) należąca do grafu G do
4:          relax(u, v, w)
5:      end for
6:  end for
7:  for każda krawędź (u, v) należąca do grafu G do
8:      if d_s[v] > d_s[u] + w[u][v] then
9:          return FALSE
10:     end if
11: end for
12: return TRUE
```



Zestaw 4, zadanie 4

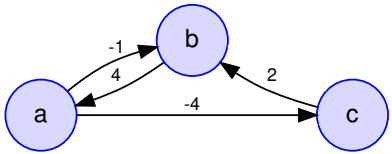
Zaimplementować algorytm Johnsona do szukania odległości pomiędzy wszystkimi parami wierzchołków na ważonym grafie skierowanym.

Algorytm Johnsona

Algorithm: `johnson`(G, w)

```
1:  G' ← add_s(G)
2:  if bellman_ford(G', w, s) = FALSE then
3:      ERROR
4:  else
5:      for każdy wierzchołek v należący do G' do
6:          h[v] ← d_s[v]
7:      end for
8:      for każda krawędź (u, v) należąca do grafu G' do
9:          ŵ[u][v] ← w[u][v] + h[u] − h[v]
10:     end for
11:     Utwórz macierz D rozmiaru n × n
12:     for każdy wierzchołek u należący do G do
13:         dijkstra(G, ŵ, u)
14:         for każdy wierzchołek v należący do G do
15:             D[u][v] ← d̂_u[v] − h[u] + h[v]
16:         end for
17:     end for
18:     return D
19: end if
```

Graf G, wagi w

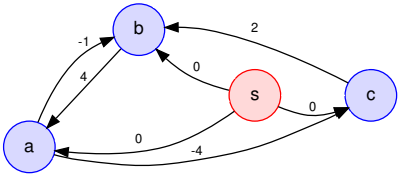


Algorytm Johnsona

Algorithm: johnson(G, w)

```
1:  G' ← add_s(G)
2:  if bellman_ford(G', w, s) = FALSE then
3:      ERROR
4:  else
5:      for każdy wierzchołek v należący do G' do
6:          h[v] ← d_s[v]
7:      end for
8:      for każda krawędź (u, v) należąca do grafu G' do
9:          ŵ[u][v] ← w[u][v] + h[u] − h[v]
10:     end for
11:     Utwórz macierz D rozmiaru n × n
12:     for każdy wierzchołek u należący do G do
13:         dijkstra(G, ŵ, u)
14:         for każdy wierzchołek v należący do G do
15:             D[u][v] ← d̂_u[v] − h[u] + h[v]
16:         end for
17:     end for
18:     return D
19: end if
```

Graf G', wagi w



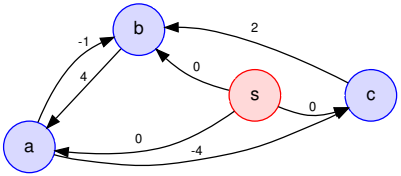
- Dodanie wierzchołka s.

Algorytm Johnsona

Algorithm: `johnson`(G, w)

```
1:  G' ← add_s(G)
2:  if bellman_ford(G', w, s) = FALSE then
3:      ERROR
4:  else
5:      for każdy wierzchołek v należący do G' do
6:          h[v] ← d_s[v]
7:      end for
8:      for każda krawędź (u, v) należąca do grafu G' do
9:          ŵ[u][v] ← w[u][v] + h[u] − h[v]
10:     end for
11:     Utwórz macierz D rozmiaru n × n
12:     for każdy wierzchołek u należący do G do
13:         dijkstra(G, ŵ, u)
14:         for każdy wierzchołek v należący do G do
15:             D[u][v] ← d̂_u[v] − h[u] + h[v]
16:         end for
17:     end for
18:     return D
19: end if
```

Graf G', wagi w



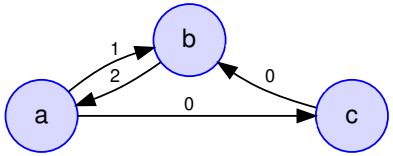
- Dodanie wierzchołka s.
- Wykluczenie **cyklu o ujemnej sumie wag** (byłby osiągalny z s).

Algorytm Johnsona

Algorithm: johnson(G, w)

```
1:  G' ← add_s(G)
2:  if bellman_ford(G', w, s) = FALSE then
3:      ERROR
4:  else
5:      for każdy wierzchołek v należący do G' do
6:          h[v] ← d_s[v]
7:      end for
8:      for każda krawędź (u, v) należąca do grafu G' do
9:          ŵ[u][v] ← w[u][v] + h[u] − h[v]
10:     end for
11:     Utwórz macierz D rozmiaru n × n
12:     for każdy wierzchołek u należący do G do
13:         dijkstra(G, ŵ, u)
14:         for każdy wierzchołek v należący do G do
15:             D[u][v] ← d̂_u[v] − h[u] + h[v]
16:         end for
17:     end for
18:     return D
19: end if
```

Graf G, wagi $\widehat{w}$



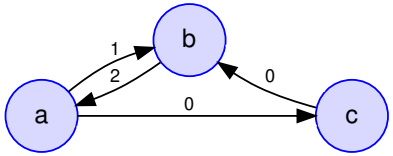
- Dodanie wierzchołka s.
- Wykluczenie **cyklu o ujemnej sumie wag** (byłby osiągalny z s).
- **Przeskalowanie wag** do nieujemnych.

Algorytm Johnsona

Algorithm: johnson(G, w)

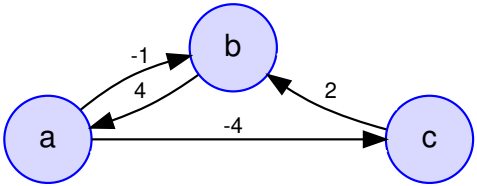
```
1:  G' ← add_s(G)
2:  if bellman_ford(G', w, s) = FALSE  then
3:      ERROR
4:  else
5:      for każdy wierzchołek v należący do G' do
6:          h[v] ← d_s[v]
7:      end for
8:      for każda krawędź (u, v) należąca do grafu G' do
9:          ŵ[u][v] ← w[u][v] + h[u] − h[v]
10:     end for
11:     Utwórz macierz D rozmiaru n × n
12:     for każdy wierzchołek u należący do G do
13:         dijkstra(G, ŵ, u)
14:         for każdy wierzchołek v należący do G do
15:             D[u][v] ← d̂_u[v] − h[u] + h[v]
16:         end for
17:     end for
18:     return D
19: end if
```

Graf G, wagi $\widehat{w}$



- Dodanie wierzchołka s.
- Wykluczenie **cyklu o ujemnej sumie wag** (byłby osiągalny z s).
- **Przeskalowanie wag** do nieujemnych.
- n razy algorytm **Dijkstry** + **powrotne przeskalowanie** wynikowych długości.

Algorytm Johnsona – przykład działania

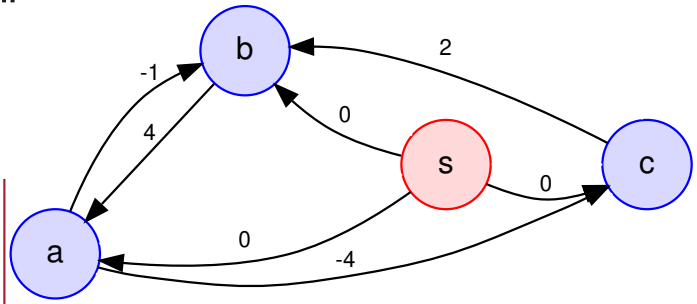


Graf G, wagi w

johnson(G, w)

- $G' \leftarrow$ add_s(G)

Algorytm Johnsona – przykład działania

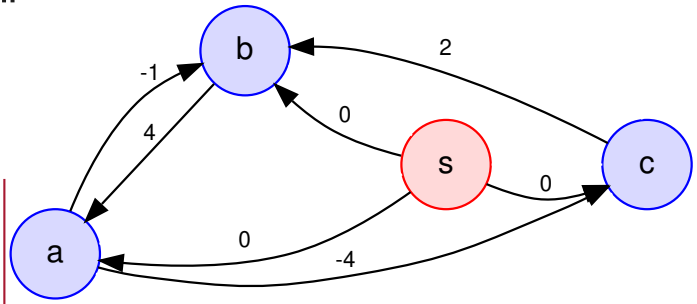


Graf $G' \leftarrow \texttt{add\_s}(G)$, wagi w

johnson(G, w)

- $G' \leftarrow \texttt{add\_s}(G)$

Algorytm Johnsona – przykład działania



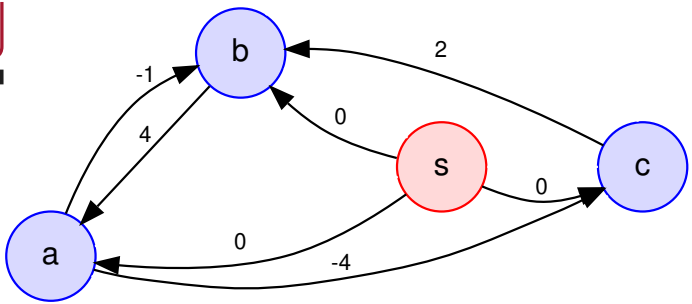
Graf $G' \leftarrow \texttt{add\_s}(G)$, wagi w

johnson(G, w)

- $G' \leftarrow \texttt{add\_s}(G)$
- bellman_ford(G', w, s)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf $G' \leftarrow$ add_s(G), wagi w

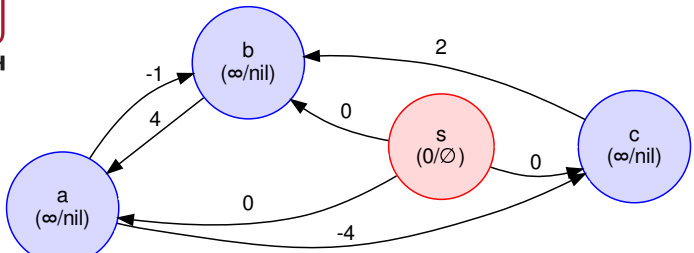
Algorithm: relax(u, v, w)

```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

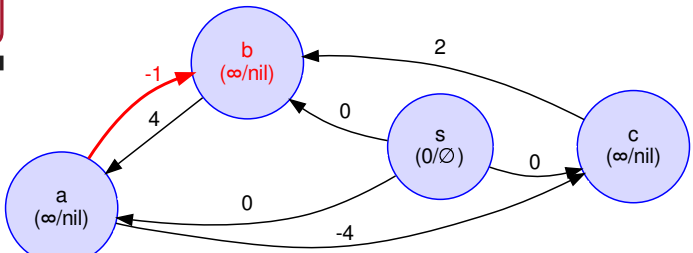
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- init(G', s)
- 1. iteracja relaksacji:
  - relax(a, b)
  - relax(a, c)
  - relax(b, a)
  - relax(c, b)
  - relax(s, a)
  - relax(s, b)
  - relax(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

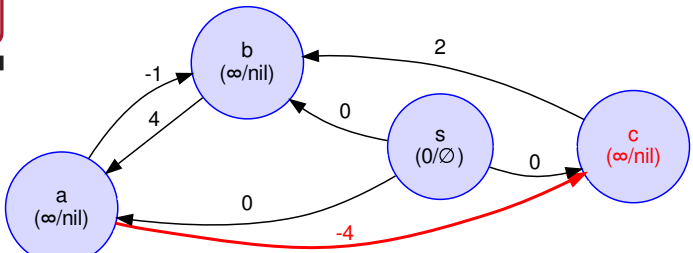
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

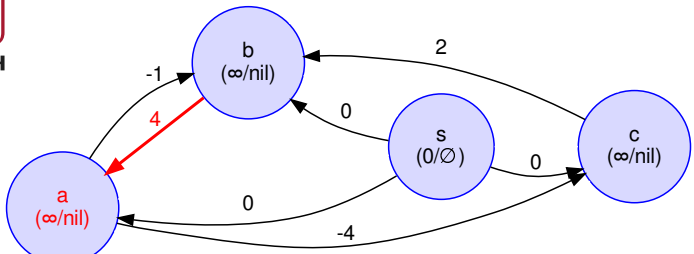
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

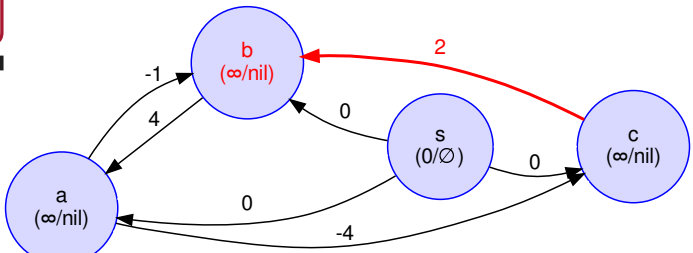
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

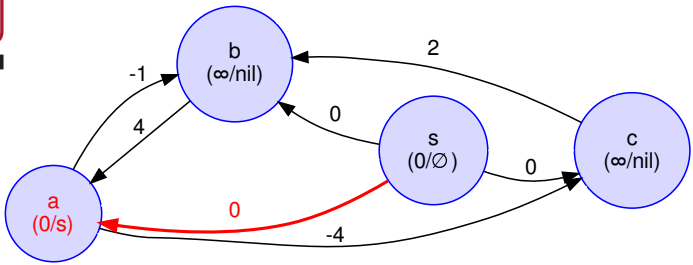
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

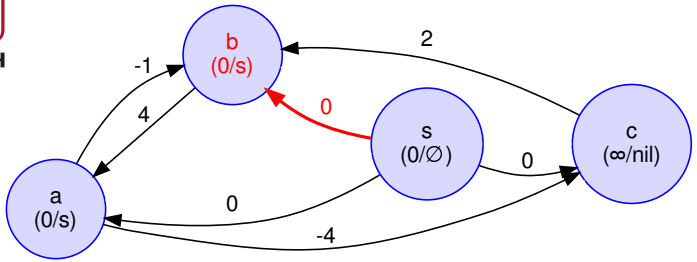
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

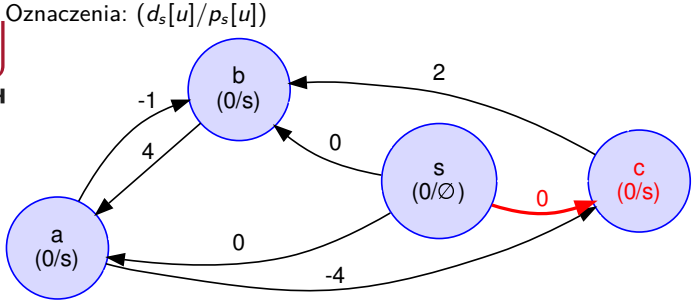
Algorithm: `relax`(u, v, w)

```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania



Graf G', wagi w

Algorithm: `relax`(u, v, w)

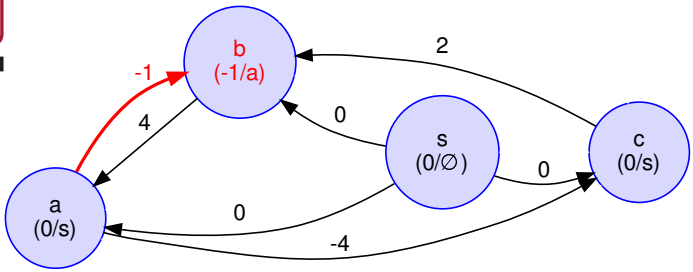
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **1. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

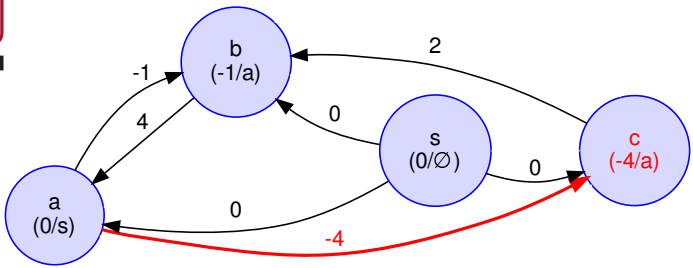
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

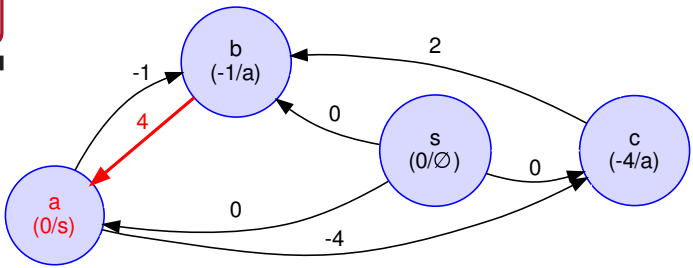
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

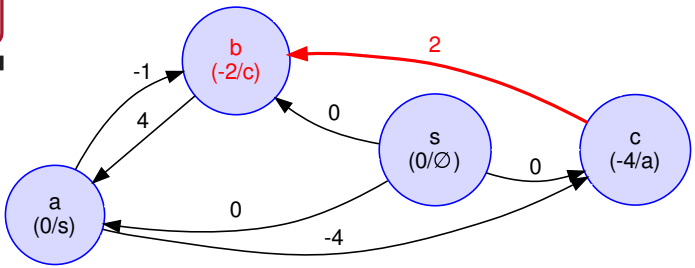
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

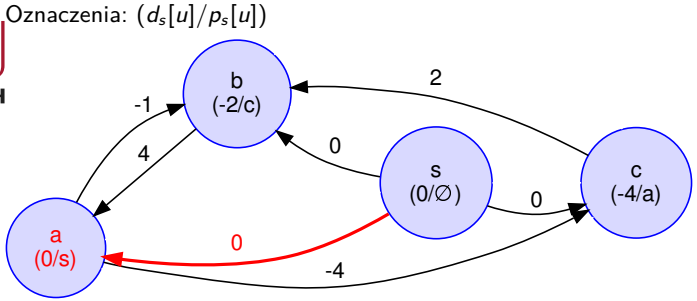
Algorithm: `relax`(u, v, w)

```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania



Graf G', wagi w

Algorithm: `relax`(u, v, w)

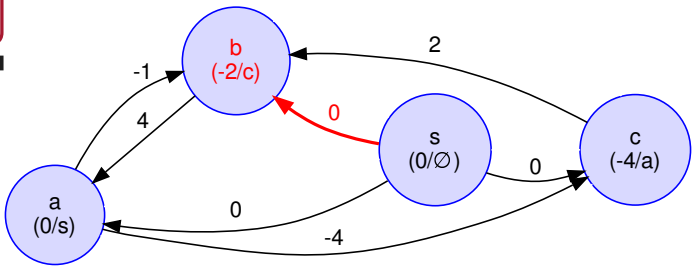
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

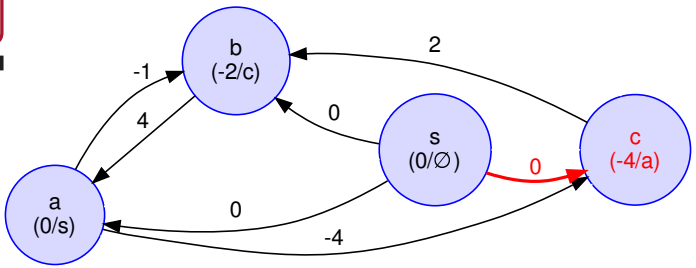
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

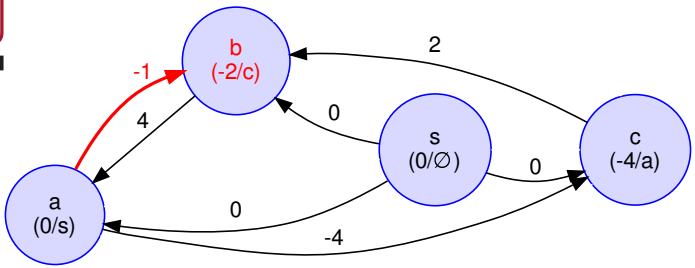
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **2. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

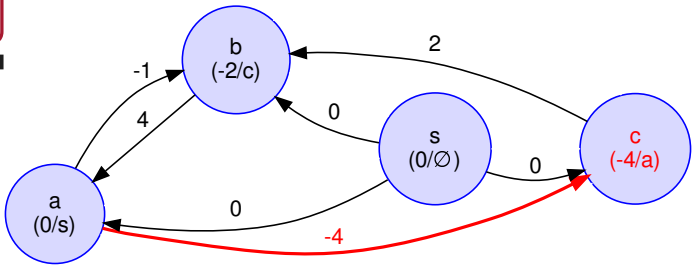
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

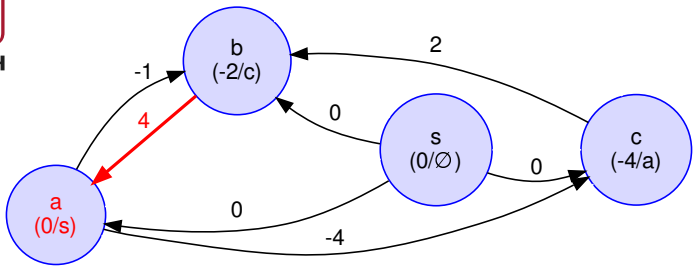
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

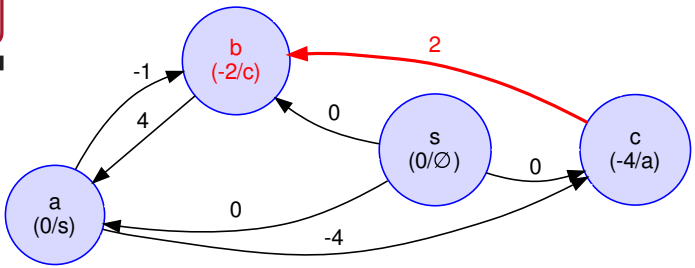
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

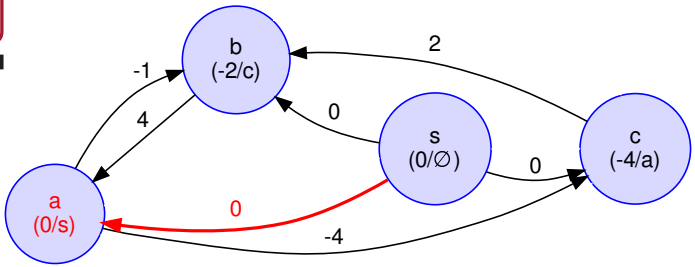
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

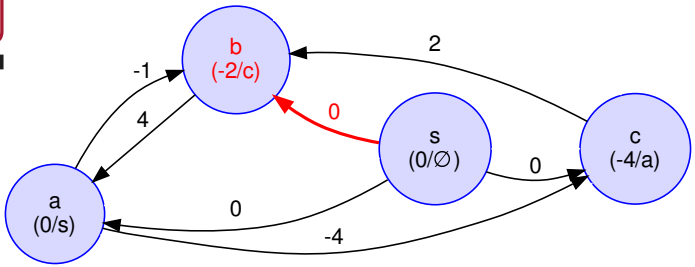
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

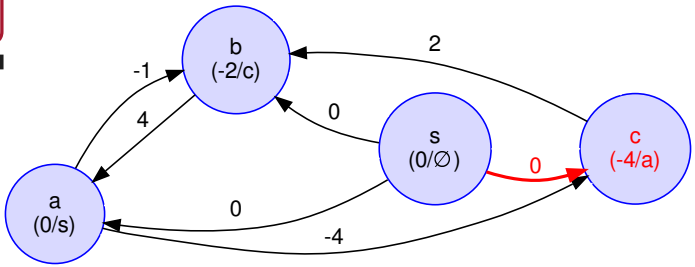
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

Algorithm: `relax`(u, v, w)

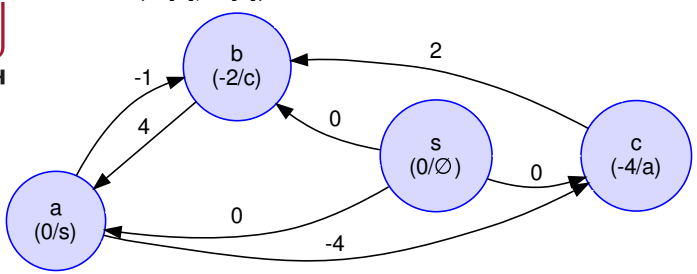
```
1: if d_s[v] > d_s[u] + w[u][v then
2:     d_s[v] ← d_s[u] + w[u][v]
3:     p_s[v] ← u
4: end if
```

bellman_ford(G', w, s)

- `init`(G', s)
- **3. iteracja relaksacji:**
  - `relax`(a, b)
  - `relax`(a, c)
  - `relax`(b, a)
  - `relax`(c, b)
  - `relax`(s, a)
  - `relax`(s, b)
  - `relax`(s, c)

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

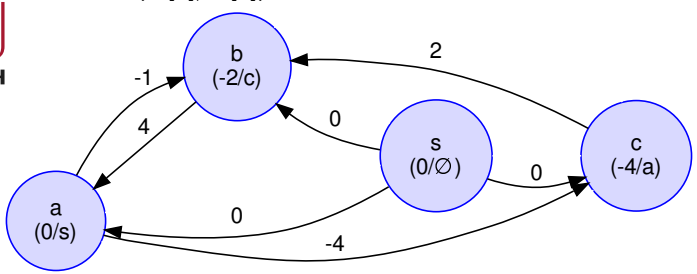
```
...
 7: for każda krawędź (u, v) należąca do grafu G do
 8:     if d_s[v] > d_s[u] + w[u][v]
 9:         return FALSE
10:     end if
11: end for
...
```

bellman_ford(G', w, s)

- **Uwaga**: to nie koniec algorytmu Bellmana Forda!
- Dla każdej z krawędzi następuje test – warunek jak w relaksacji.
- Jeśli warunek spełniony $\Rightarrow$ w grafie jest ujemny cykl!
- Czy u nas jest ujemny cykl...?

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

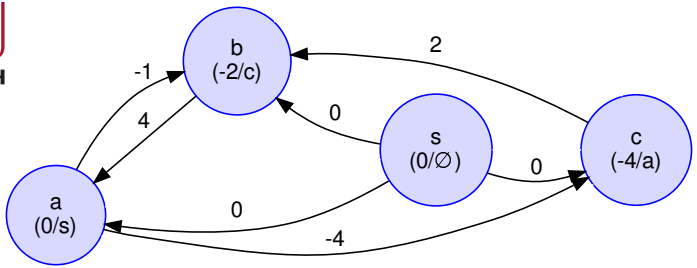
```
...
 7: for każda krawędź (u, v) należąca do grafu G do
 8:     if d_s[v] > d_s[u] + w[u][v]
 9:         return FALSE
10:     end if
11: end for
...
```

bellman_ford(G', w, s)

- **Uwaga**: to nie koniec algorytmu Bellmana Forda!
- Dla każdej z krawędzi następuje test – warunek jak w relaksacji.
- Jeśli warunek spełniony $\Rightarrow$ w grafie jest ujemny cykl!
- Czy u nas jest ujemny cykl...? **Nie.**

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

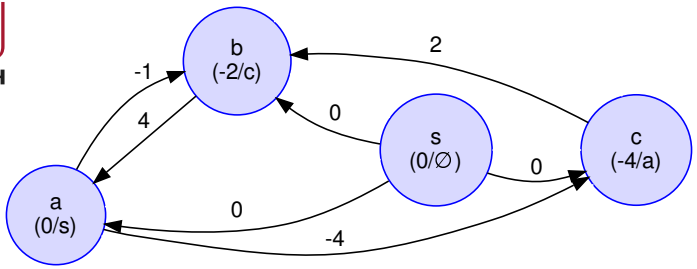
```
...
 5: for każdy wierzchołek v należący do G' do
 6:     h[v] ← d_s[v]
 7: end for
 8: for każda krawędź (u, v) należąca do grafu G' do
 9:     ŵ[u][v] ← w[u][v] + h[u] − h[v]
10: end for
...
```

johnson(G, w) – cd.

- Z Bellmana-Forda: mamy d_s.

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

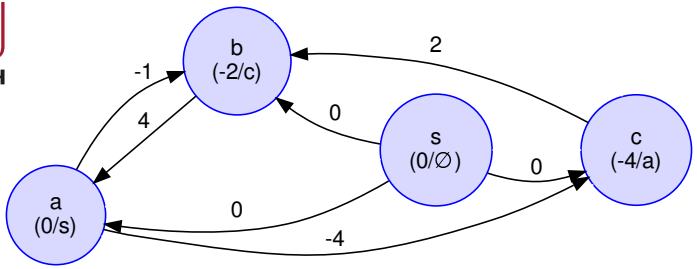
```
...
 5: for każdy wierzchołek v należący do G' do
 6:     h[v] ← d_s[v]
 7: end for
 8: for każda krawędź (u, v) należąca do grafu G' do
 9:     ŵ[u][v] ← w[u][v] + h[u] − h[v]
10: end for
...
```

johnson(G, w) – cd.

- Z Bellmana-Forda: mamy d_s.
  - $h[a] = 0$,
  - $h[b] = -2$,
  - $h[c] = -4$,
  - $h[s] = 0$.

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$



Graf G', wagi w

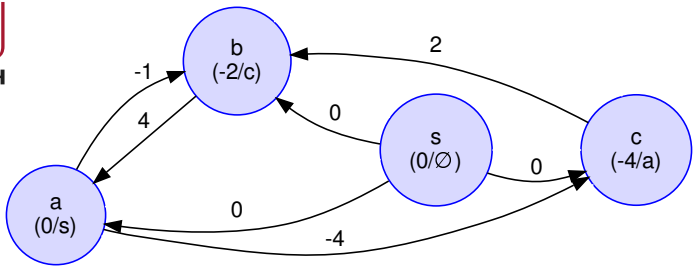
```
...
 5: for każdy wierzchołek v należący do G' do
 6:     h[v] ← d_s[v]
 7: end for
 8: for każda krawędź (u, v) należąca do grafu G' do
 9:     ŵ[u][v] ← w[u][v] + h[u] − h[v]
10: end for
...
```

johnson(G, w) – cd.

- Z Bellmana-Forda: mamy d_s.
  - $h[a] = 0$,
  - $h[b] = -2$,
  - $h[c] = -4$,
  - $h[s] = 0$.
- **Przeskalowanie wag** w do nieujemnych wag $\widehat{w}$.

Algorytm Johnsona – przykład działania

Oznaczenia: $(d_s[u]/p_s[u])$

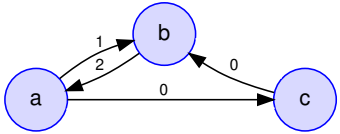


Graf G', wagi w

```
...
 5: for każdy wierzchołek v należący do G' do
 6:     h[v] ← d_s[v]
 7: end for
 8: for każda krawędź (u, v) należąca do grafu G' do
 9:     ŵ[u][v] ← w[u][v] + h[u] − h[v]
10: end for
...
```

johnson(G, w) – cd.

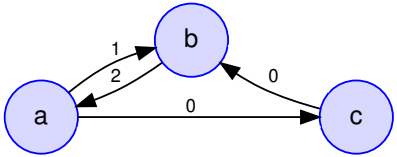
- Z Bellmana-Forda: mamy d_s.
  - $h[a] = 0$,
  - $h[b] = -2$,
  - $h[c] = -4$,
  - $h[s] = 0$.
- **Przeskalowanie wag** w do nieujemnych wag $\widehat{w}$.



Graf G, wagi $\widehat{w}$

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$



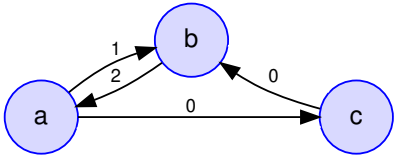

```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

- Mamy:
  $h[a] = 0$, $h[b] = -2$, $h[c] = -4$.
- dijkstra$(G, \widehat{w}, a)$
- dijkstra$(G, \widehat{w}, b)$
- dijkstra$(G, \widehat{w}, c)$

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

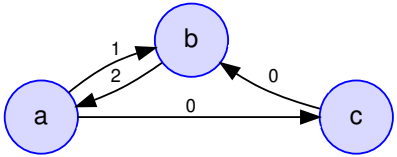
- Mamy:
  $h[a] = 0$, $h[b] = -2$, $h[c] = -4$.
- dijkstra$(G, \widehat{w}, a)$
  - $\widehat{d}_a[a] = 0$,
  - $\widehat{d}_a[b] = 0$,
  - $\widehat{d}_a[c] = 0$.
- dijkstra$(G, \widehat{w}, b)$
- dijkstra$(G, \widehat{w}, c)$

$D =$

	a	b	c
a			
b			
c			

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

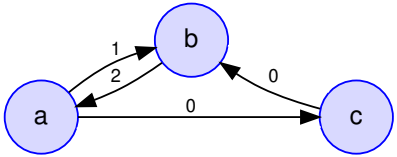
- Mamy:
  $h[a] = 0,\ h[b] = -2,\ h[c] = -4.$
- dijkstra$(G, \widehat{w}, a)$
  - $\widehat{d}_a[a] = 0,$
  - $\widehat{d}_a[b] = 0,$
  - $\widehat{d}_a[c] = 0.$
- dijkstra$(G, \widehat{w}, b)$
- dijkstra$(G, \widehat{w}, c)$

$D =$

	a	b	c
a	0	-2	-4
b			
c			

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

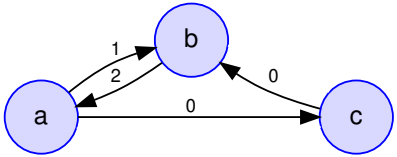
- Mamy:
  $h[a] = 0,\ h[b] = -2,\ h[c] = -4.$
- dijkstra$(G, \widehat{w}, a)$
- dijkstra$(G, \widehat{w}, b)$
  - $\widehat{d}_b[a] = 2,$
  - $\widehat{d}_b[b] = 0,$
  - $\widehat{d}_b[c] = 2.$
- dijkstra$(G, \widehat{w}, c)$

$D =$

	a	b	c
a	0	-2	-4
b			
c			

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

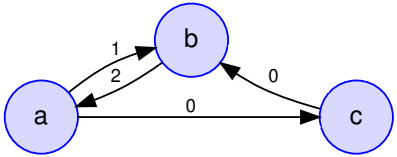
- Mamy:
  $h[a] = 0,\ h[b] = -2,\ h[c] = -4.$
- dijkstra$(G, \widehat{w}, a)$
- dijkstra$(G, \widehat{w}, b)$
  - $\widehat{d}_b[a] = 2,$
  - $\widehat{d}_b[b] = 0,$
  - $\widehat{d}_b[c] = 2.$
- dijkstra$(G, \widehat{w}, c)$

$D =$

	a	b	c
a	0	-2	-4
b	4	0	0
c			

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

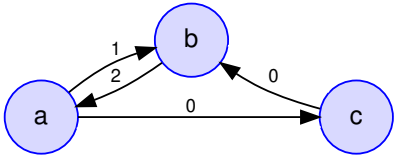
- Mamy:
  $h[a] = 0,\ h[b] = -2,\ h[c] = -4.$
- dijkstra$(G, \widehat{w}, a)$
- dijkstra$(G, \widehat{w}, b)$
- dijkstra$(G, \widehat{w}, c)$
  - $\widehat{d}_c[a] = 2,$
  - $\widehat{d}_c[b] = 0,$
  - $\widehat{d}_c[c] = 0.$

$D =$

	a	b	c
a	0	-2	-4
b	4	0	0
c			

Algorytm Johnsona – przykład działania

Graf G, wagi $\widehat{w}$




```
...
12: for każdy wierzchołek u należący do G do
13:     dijkstra(G, ŵ, u)
14:     for każdy wierzchołek v należący do G do
15:         D[u][v] ← d̂_u[v] − h[u] + h[v]
16:     end for
17: end for
...
```

johnson(G, w) – cd.

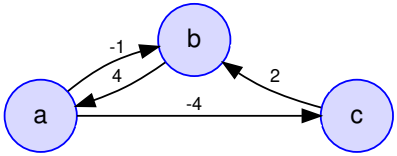
- Mamy:
  $h[a] = 0,\ h[b] = -2,\ h[c] = -4.$
- dijkstra$(G, \widehat{w}, a)$
- dijkstra$(G, \widehat{w}, b)$
- dijkstra$(G, \widehat{w}, c)$
  - $\widehat{d}_c[a] = 2,$
  - $\widehat{d}_c[b] = 0,$
  - $\widehat{d}_c[c] = 0.$

$D =$

	a	b	c
a	0	-2	-4
b	4	0	0
c	6	2	0

Algorytm Johnsona – podsumowanie

Graf G, wagi w



$D =$

	a	b	c
a	**0**	**-2**	**-4**
b	**4**	**0**	**0**
c	**6**	**2**	**0**

- Złożoność zależy od implementacji algorytmu Dijkstry: między $O(n \cdot k + n^2 \log(n))^a$ a $O(n \cdot k + n^3)^b$.

  n – liczba wierzchołków, k – liczba krawędzi.
- Prosta alternatywa: n-razy algorytm Bellmana-Forda, złożoność $O(n^2 \cdot k)$:(
- **Przeskalowanie wag nie ma wpływu na najkrótsze ścieżki** (tj. na kolejność krawędzi w ścieżce), zmienia tylko ich długość.

[a] Implementacja z kopcami Fibonacciego. Przy binarnych: $O(n \cdot k + n \cdot k \log(n))$.
[b] Implementacja z tablicą.

Dla zainteresowanych

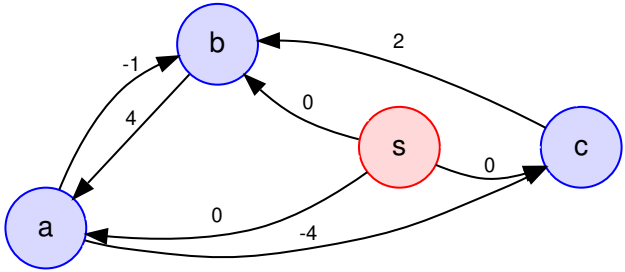
1. Czy algorytmu Dijkstry nie można stosować przy ujemnych wagach tylko ze względu na niemożliwość zidentyfikowania ujemnego cyklu?

Dla zainteresowanych

1. Czy algorytmu Dijkstry nie można stosować przy ujemnych wagach tylko ze względu na niemożliwość zidentyfikowania ujemnego cyklu?

Nie. Przykład:

- Mamy dany graf, szukamy najkrótszych ścieżek od s.

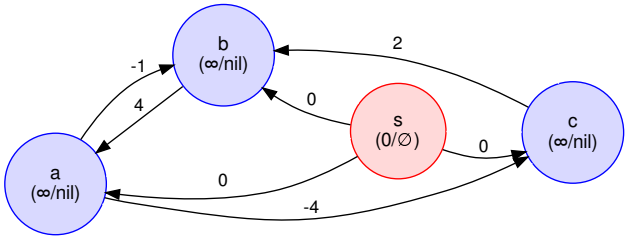


Dla zainteresowanych

1. Czy algorytmu Dijkstry nie można stosować przy ujemnych wagach tylko ze względu na niemożliwość zidentyfikowania ujemnego cyklu?

Nie. Przykład:

- Mamy dany graf, szukamy najkrótszych ścieżek od s.
- Inicjalizacja tablic d_s/p_s.

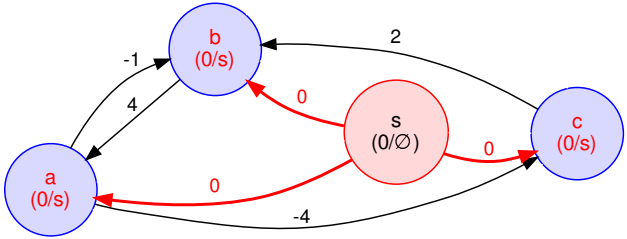


Dla zainteresowanych

1. Czy algorytmu Dijkstry nie można stosować przy ujemnych wagach tylko ze względu na niemożliwość zidentyfikowania ujemnego cyklu?

Nie. Przykład:

- Mamy dany graf, szukamy najkrótszych ścieżek od s.
- Inicjalizacja tablic d_s/p_s.
- Wierzchołek gotowy: $s \Rightarrow$ relaksacja (s, a), (s, b), (s, c).

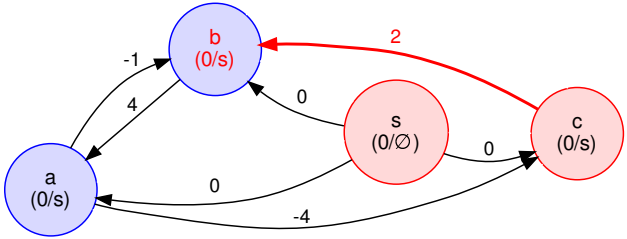


Dla zainteresowanych

1. Czy algorytmu Dijkstry nie można stosować przy ujemnych wagach tylko ze względu na niemożliwość zidentyfikowania ujemnego cyklu?

Nie. Przykład:

- Mamy dany graf, szukamy najkrótszych ścieżek od s.
- Inicjalizacja tablic d_s/p_s.
- Wierzchołek gotowy: $s \Rightarrow$ relaksacja (s, a), (s, b), (s, c).
- Wierzchołek gotowy: **dowolność wyboru**, skoro mają równe d_s! Weźmy c, więc uznajemy, że $d_s[c] = 0$, czyli ścieżka $s - c$ jest najkrótsza. **Błąd!**



Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dowód nie wprost

- Założenie: $\widehat{w}(p)$ nie jest minimalne $\Rightarrow \exists_{p'}$: $\widehat{w}(p')$ – minimalne.

$$\widehat{w}(p') < \widehat{w}(p)$$

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dowód nie wprost

- Założenie: $\widehat{w}(p)$ nie jest minimalne $\Rightarrow \exists_{p'}$: $\widehat{w}(p')$ – minimalne.

$$\widehat{w}(p') < \widehat{w}(p)$$

$$\widehat{w}(p') = w(p') + h(u) - h(v)$$
$$\widehat{w}(p) = w(p) + h(u) - h(v)$$

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dowód nie wprost

- Założenie: $\widehat{w}(p)$ nie jest minimalne $\Rightarrow \exists_{p'}$: $\widehat{w}(p')$ – minimalne.

$$\widehat{w}(p') < \widehat{w}(p)$$

$$\widehat{w}(p') = w(p') + h(u) - h(v)$$
$$\widehat{w}(p) = w(p) + h(u) - h(v)$$

$$w(p') + h(u) - h(v) < w(p) + h(u) - h(v)$$

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dowód nie wprost

- Założenie: $\widehat{w}(p)$ nie jest minimalne $\Rightarrow \exists_{p'}$: $\widehat{w}(p')$ – minimalne.

$$\widehat{w}(p') < \widehat{w}(p)$$

$$\widehat{w}(p') = w(p') + h(u) - h(v)$$
$$\widehat{w}(p) = w(p) + h(u) - h(v)$$

$$w(p') + \cancel{h(u)} - \cancel{h(v)} < w(p) + \cancel{h(u)} - \cancel{h(v)}$$

$$w(p') < w(p)$$

Dla zainteresowanych

2. Dlaczego przeskalowanie wag nie zmienia najkrótszych ścieżek?

Oznaczenia

- w – wagi przed przeskalowaniem.
- $\widehat{w}$ – wagi po przeskalowaniu.
- Założenie: p – najkrótsza ścieżka z u do v przy wagach w, tj. $w(p) = w(u \to v)$ – minimalne.
- Potrzebujemy dowieść, że $\widehat{w}(p)$ też minimalne (przy wagach $\widehat{w}$).

Dowód nie wprost

- Założenie: $\widehat{w}(p)$ nie jest minimalne $\Rightarrow \exists_{p'}$: $\widehat{w}(p')$ – minimalne.

$$\widehat{w}(p') < \widehat{w}(p)$$

$$\widehat{w}(p') = w(p') + h(u) - h(v)$$
$$\widehat{w}(p) = w(p) + h(u) - h(v)$$

$$w(p') + \cancel{h(u)} - \cancel{h(v)} < w(p) + \cancel{h(u)} - \cancel{h(v)}$$

$$\cancel{w(p') < w(p)}$$

- **Sprzeczność!** Wniosek: skoro p najkrótsza przy wagach w, to przy wagach po przeskalowaniu też nie istnieje krótsza ścieżka.