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

Grafy i ich zastosowania

Zestaw 3

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

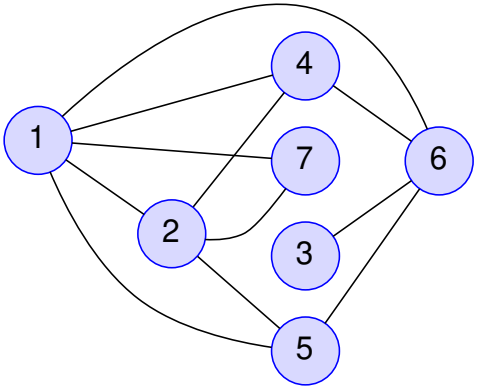
AGH

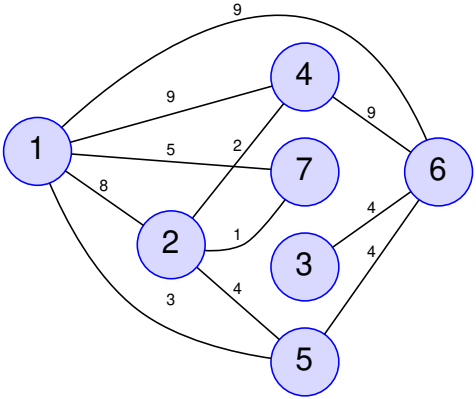
Zestaw 3, zadanie 1

Korzystając z programów z poprzednich zestawów, wygenerować spójny graf losowy. Przypisać każdej krawędzi tego grafu losową wagę będącą liczbą naturalną z zakresu od 1 do 10.

Losowanie grafu

- Losujemy graf spójny, **prosty** oraz **dodatnie wagi** dla wszystkich krawędzi.



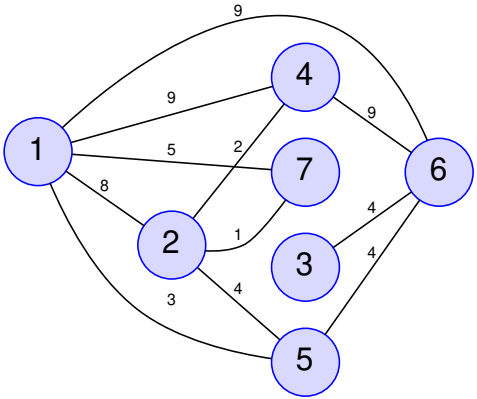


AGH

Zestaw 3, zadanie 2

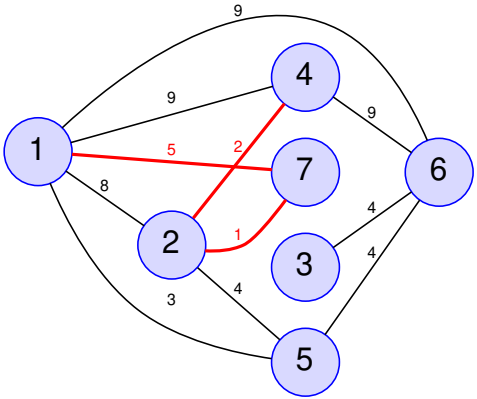
Zaimplementować algorytm Dijkstry do znajdowania najkrótszych ścieżek od zadanego wierzchołka do pozostałych wierzchołków i zastosować go do grafu z zadania pierwszego, w którym wagi krawędzi interpretowane są jako odległości wierzchołków. Wypisać wszystkie najkrótsze ścieżki od danego wierzchołka i ich długości.

Poszukiwanie najkrótszych ścieżek



Najkrótsza ścieżka $1 \rightarrow 4$?

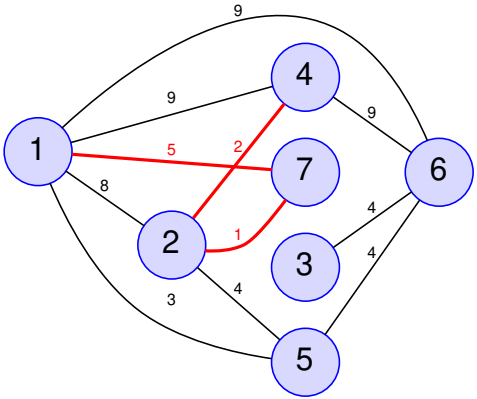
Poszukiwanie najkrótszych ścieżek



Najkrótsza ścieżka $1 \to 4$?

1 – 7 – 2 – 4

Poszukiwanie najkrótszych ścieżek



Najkrótsza ścieżka $1 \to 4$?

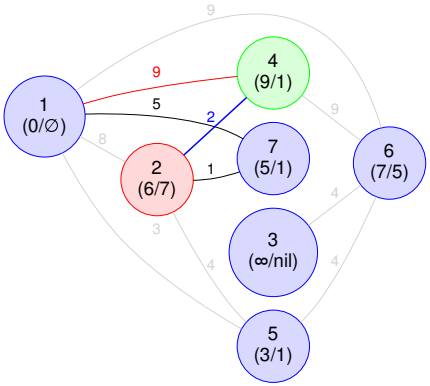
1 – 7 – 2 – 4

- Różne algorytmy wykorzystujące metodę **relaksacji** (osłabiania ograniczeń).
- Wspólne: inicjalizacja ($\texttt{init}(G, s)$) oraz relaksacja ($\texttt{relax}(u, v, w)$).
- Różnice: **kolejność** relaksacji krawędzi i **liczba ich powtórzeń**.

Tablice d_s i p_s

- $d_s[u]$ – długość najkrótszej ścieżki $s \to u$. W trakcie algorytmu: górne oszacowanie tej długości.
- $p_s[u]$ – poprzednik wierzchołka u na najkrótszej ścieżce $s \to u$.

Poszukiwanie najkrótszych ścieżek: relaksacja



- `relax`(u, v, w) sprawdza, czy z s do v da się przejść krócej przez krawędź (u, v) (względem aktualnego oszacowania $d_s[v]$).

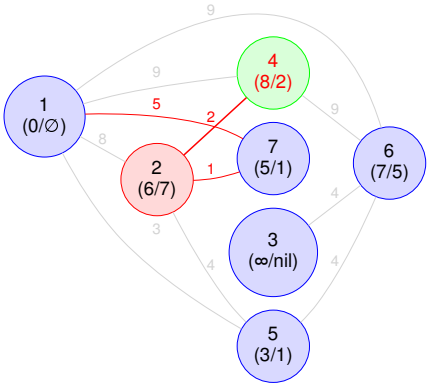
Algorithm: `init`(G, s)

```
1: for ∀ wierzchołek v ∈ G do
2:     d_s[v] ← ∞
3:     p_s[v] ←  NIL
4: end for
5: d_s[s] ← 0
```

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
```

Poszukiwanie najkrótszych ścieżek: relaksacja



- `relax`(u, v, w) sprawdza, czy z s do v da się przejść krócej przez krawędź (u, v) (względem aktualnego oszacowania $d_s[v]$).

Algorithm: `init`(G, s)

```
1: for ∀ wierzchołek v ∈ G do
2:     d_s[v] ← ∞
3:     p_s[v] ←  NIL
4: end for
5: d_s[s] ← 0
```

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
```

Poszukiwanie najkrótszych ścieżek: 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
```

- **Inicjalizacja tablic** d_s, p_s. Zbiór S: pusty („gotowe" wierzchołki).

Poszukiwanie najkrótszych ścieżek: 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
```

- **Inicjalizacja tablic** d_s, p_s. Zbiór S: pusty („gotowe" wierzchołki).
- Wybieramy wierzchołek o **najmniejszym** $d_s[u]$ – „gotowy", dodajemy do S.

Poszukiwanie najkrótszych ścieżek: 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
```

- **Inicjalizacja tablic** d_s, p_s. Zbiór S: pusty („gotowe" wierzchołki).
- Wybieramy wierzchołek o **najmniejszym** $d_s[u]$ – „gotowy", dodajemy do S.
- **Relaksacja** krawędzi do jego „niegotowych" sąsiadów.

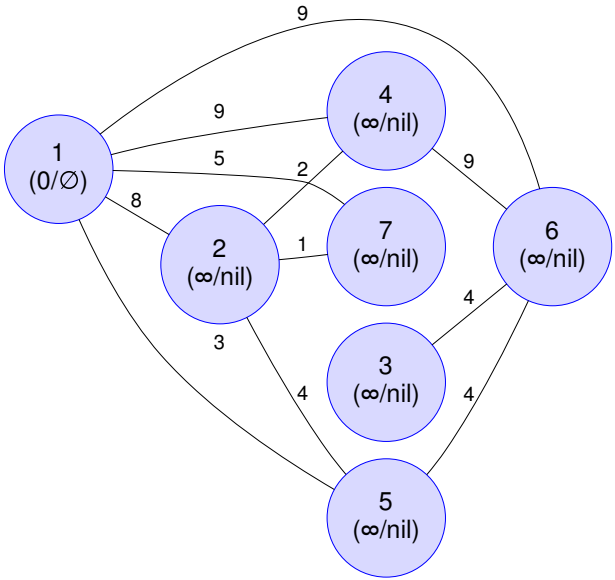
Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry

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
```

- Wierzchołki **gotowe** – bo dalsza relaksacja nie dałaby już poprawy. Długość tych najkrótszych ścieżek jest już obliczona $\Rightarrow$ tylko **jedna relaksacja** każdej krawędzi.

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



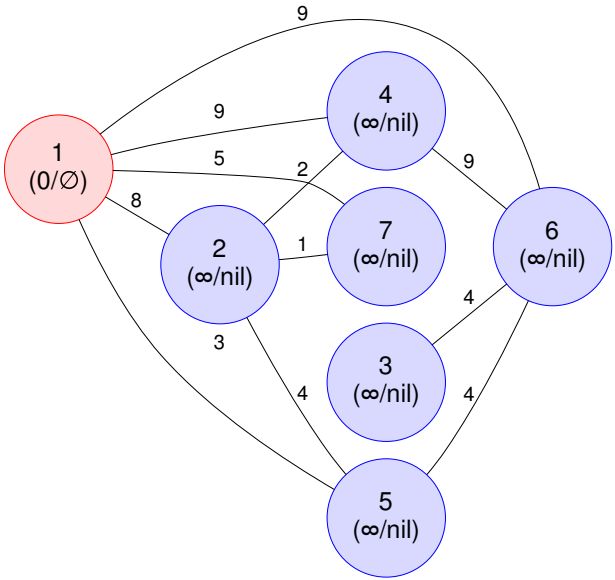
Oznaczenia: $(d_s[u]/p_s[u])$

- `init`$(G, s = 1)$
- $S = \varnothing$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



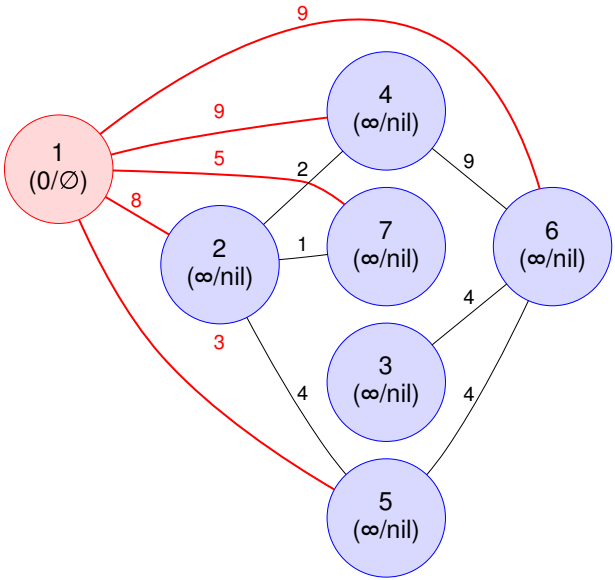
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- `relax`$(1, 2)$
- `relax`$(1, 4)$
- `relax`$(1, 5)$
- `relax`$(1, 6)$
- `relax`$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



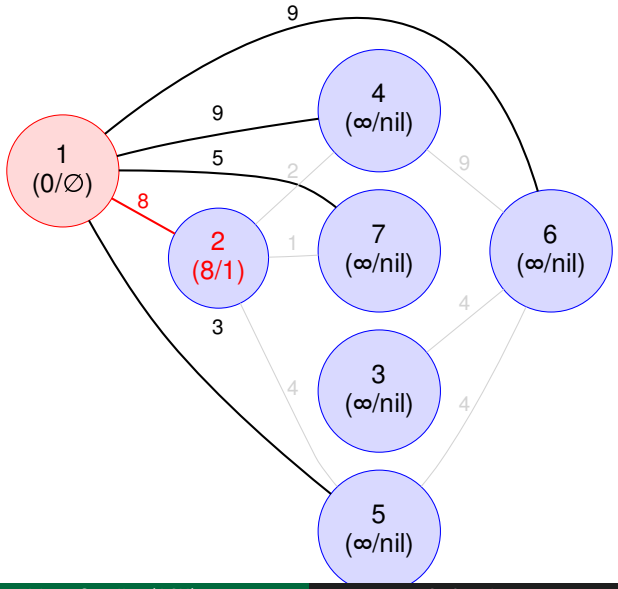
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- `relax`$(1, 2)$
- `relax`$(1, 4)$
- `relax`$(1, 5)$
- `relax`$(1, 6)$
- `relax`$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



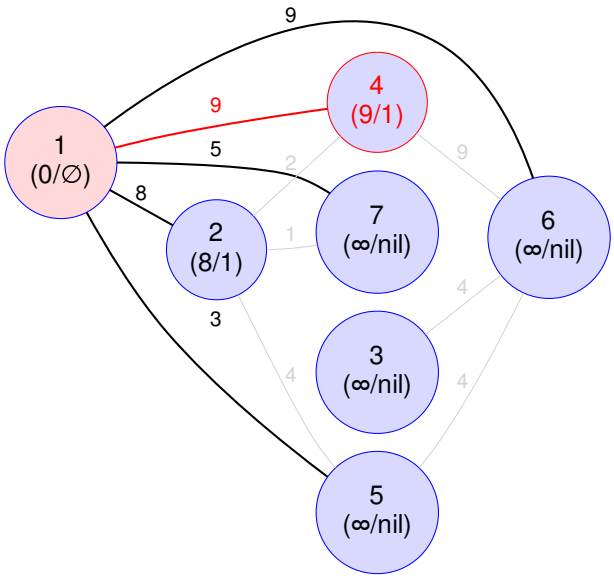
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- relax$(1, 2)$
- relax$(1, 4)$
- relax$(1, 5)$
- relax$(1, 6)$
- relax$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



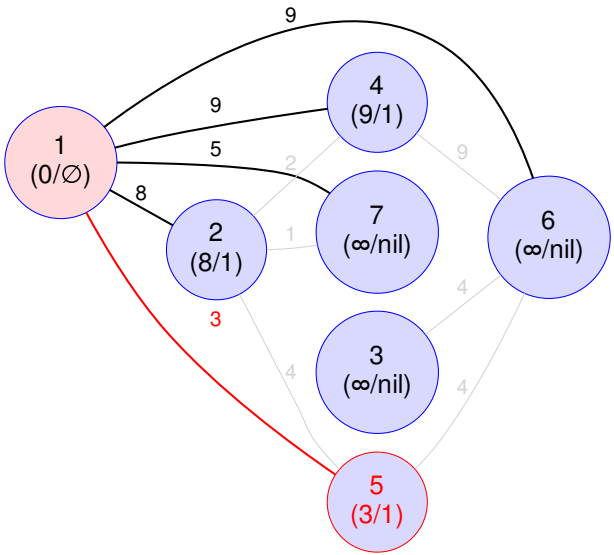
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- relax$(1, 2)$
- relax$(1, 4)$
- relax$(1, 5)$
- relax$(1, 6)$
- relax$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



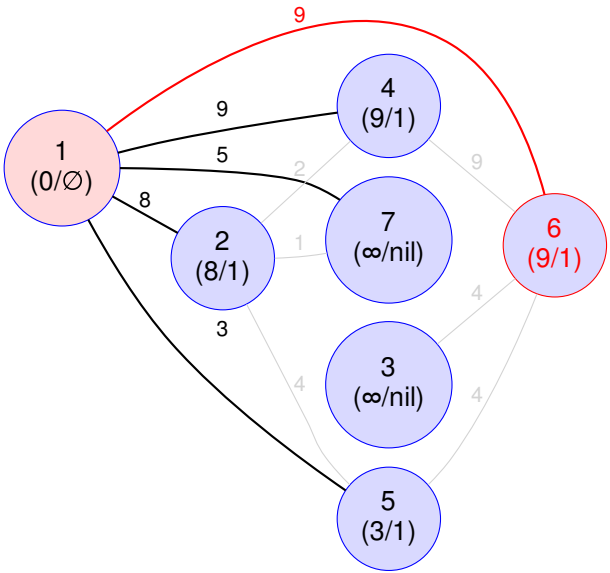
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- relax$(1, 2)$
- relax$(1, 4)$
- relax$(1, 5)$
- relax$(1, 6)$
- relax$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



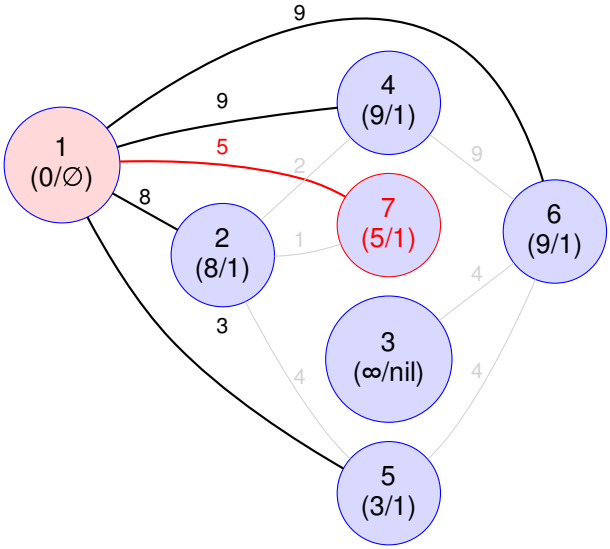
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- relax$(1, 2)$
- relax$(1, 4)$
- relax$(1, 5)$
- relax$(1, 6)$
- relax$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



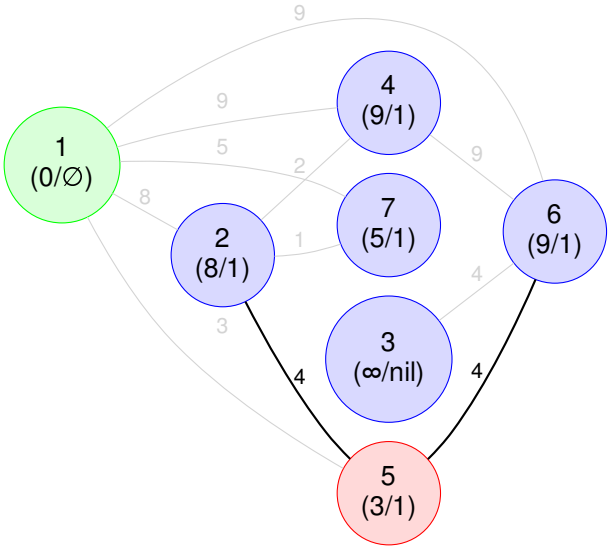
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\}$
- relax$(1, 2)$
- relax$(1, 4)$
- relax$(1, 5)$
- relax$(1, 6)$
- relax$(1, 7)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



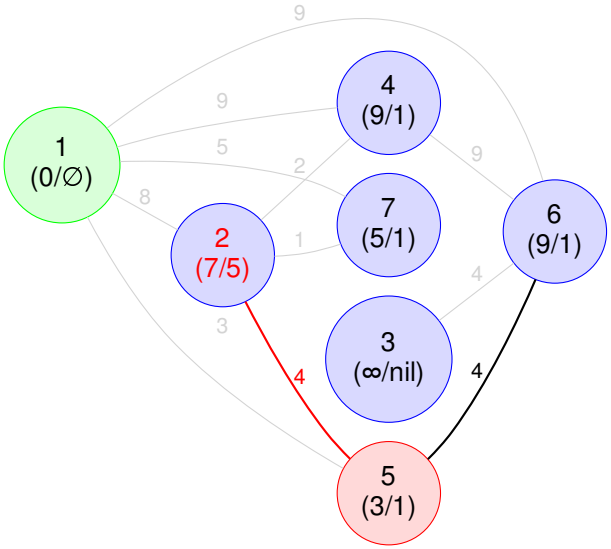
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\} \cup 5$
- `relax`$(5, 2)$
- `relax`$(5, 6)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



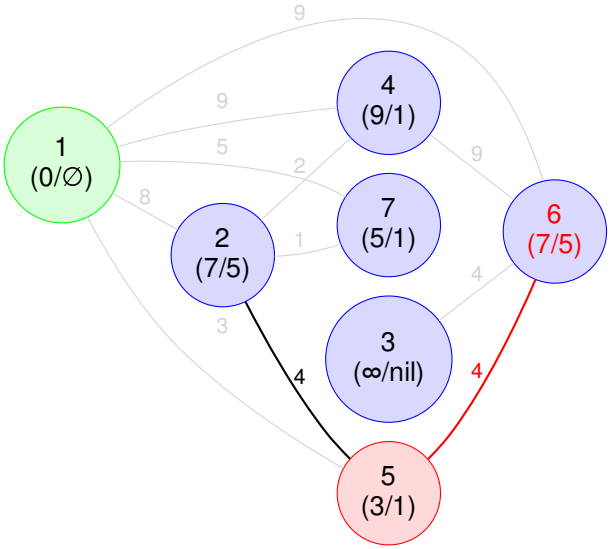
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\} \cup 5$
- relax(5, 2)
- relax(5, 6)

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



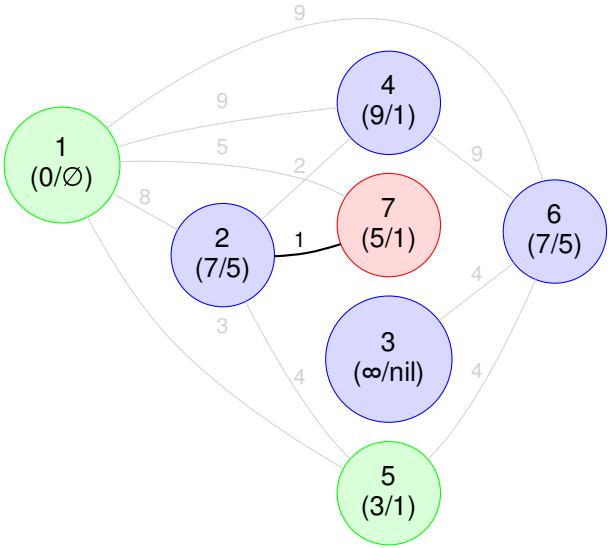
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1\} \cup 5$
- relax$(5, 2)$
- relax$(5, 6)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



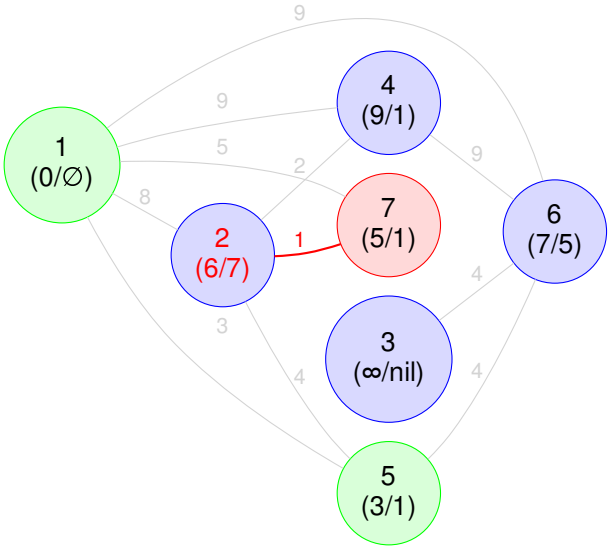
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 5\} \cup 7$
- `relax`$(7, 2)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



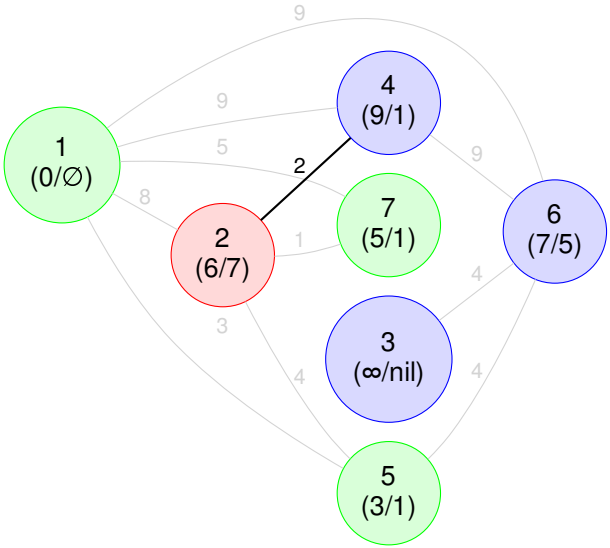
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 5\} \cup 7$
- `relax`$(7, 2)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



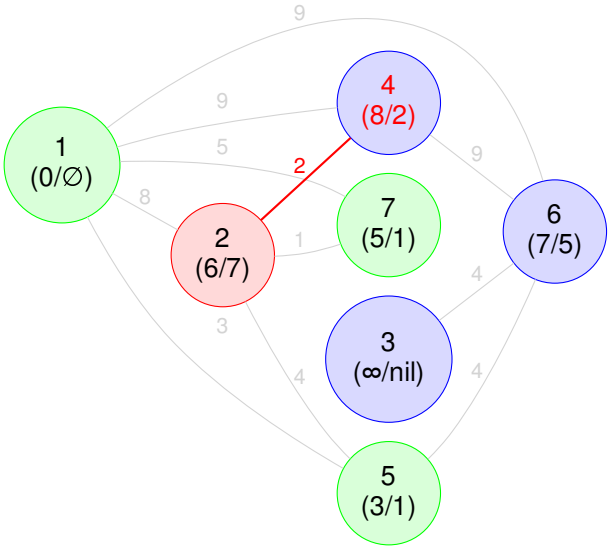
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 5, 7\} \cup 2$
- `relax`$(2, 4)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



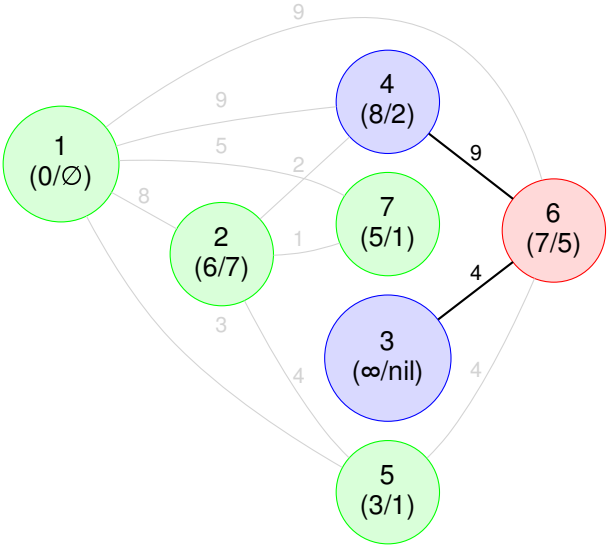
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 5, 7\} \cup 2$
- `relax`$(2, 4)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



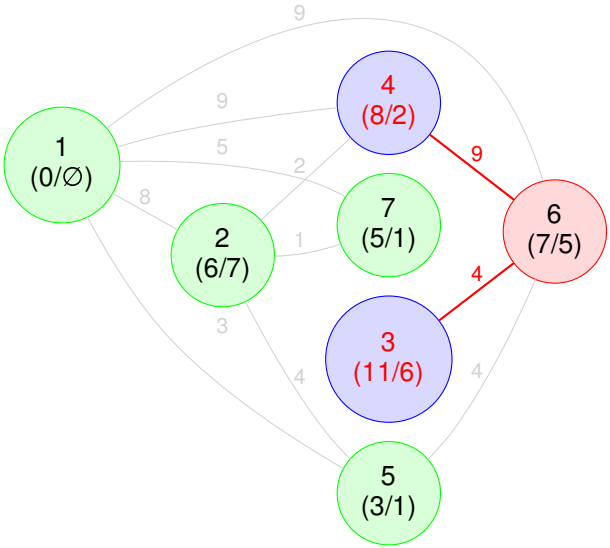
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 2, 5, 7\} \cup 6$
- `relax`$(6, 3)$
- `relax`$(6, 4)$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



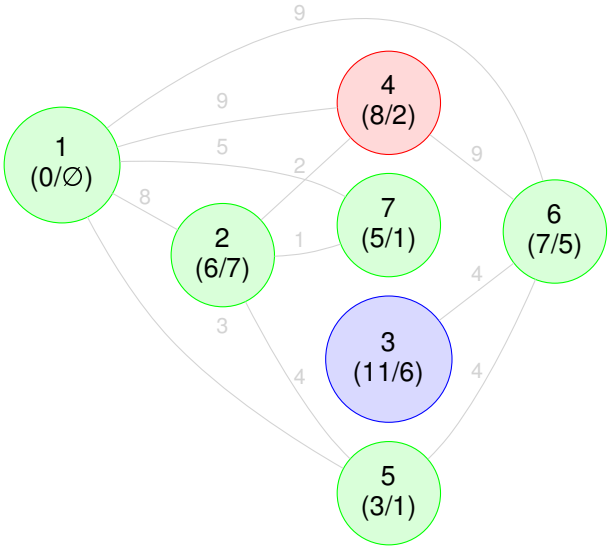
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 2, 5, 7\} \cup 6$
- relax(6, 3)
- relax(6, 4)

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



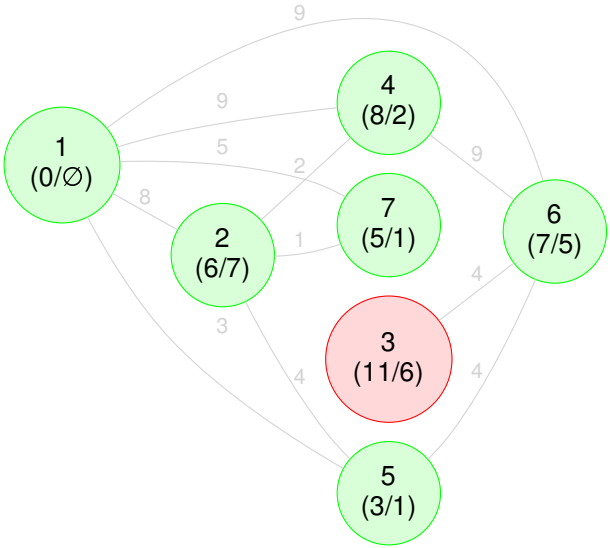
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 2, 5, 6, 7\} \cup 4$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



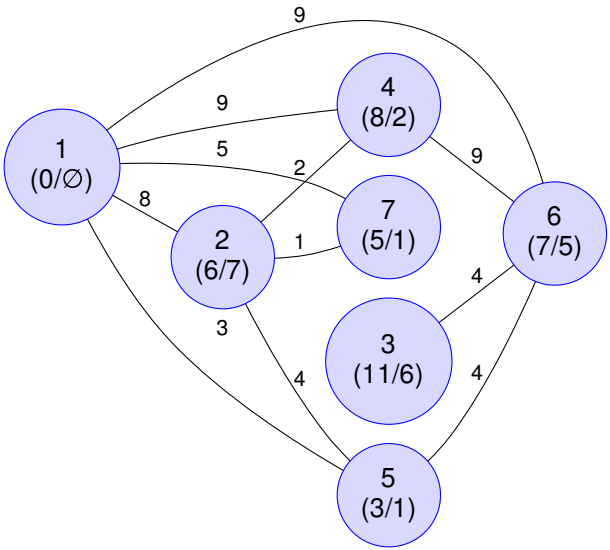
Oznaczenia: $(d_s[u]/p_s[u])$

- $S = \{1, 2, 4, 5, 6, 7\} \cup 3$

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
```

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

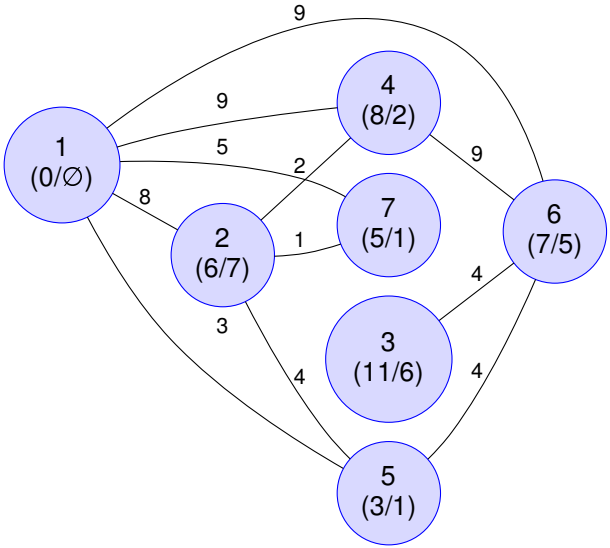


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$:

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

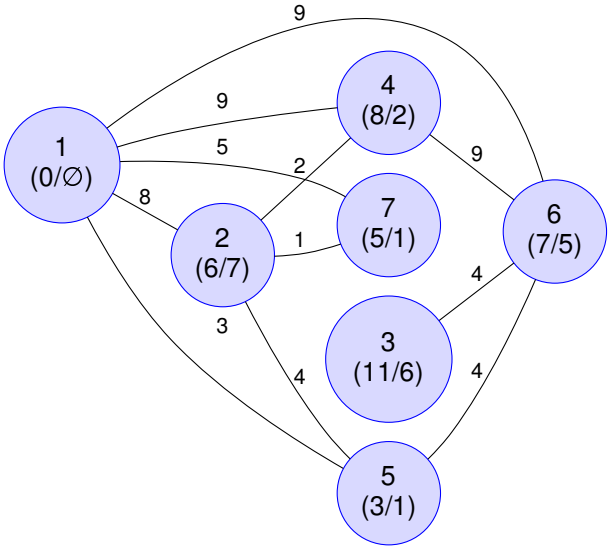


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: 2

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

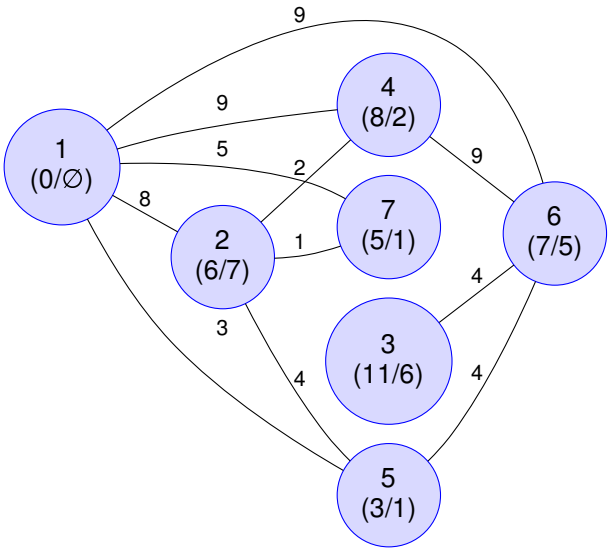


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: 7 – 2

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

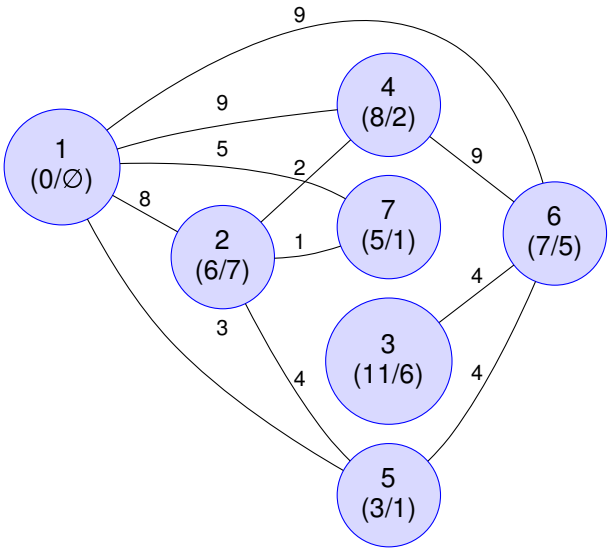


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: 1 – 7 – 2

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

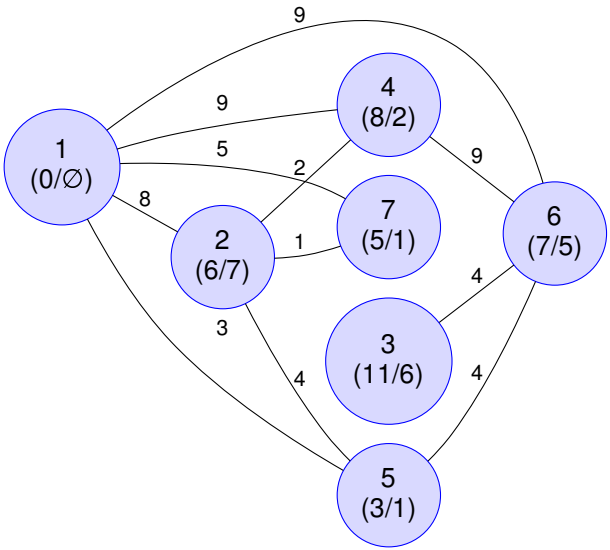


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: 3

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

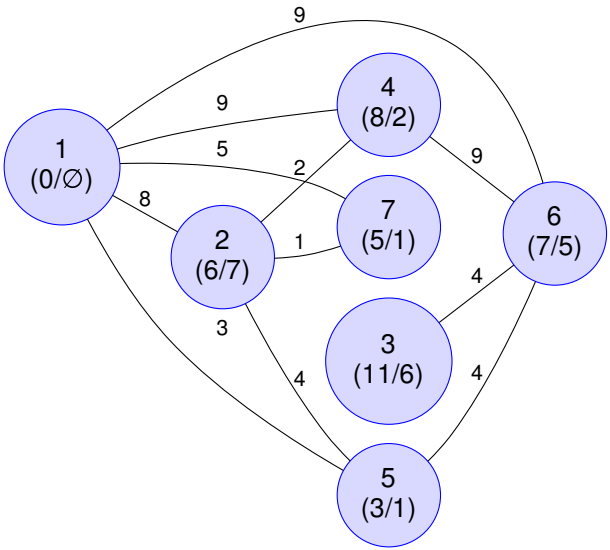


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $6 - 3$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

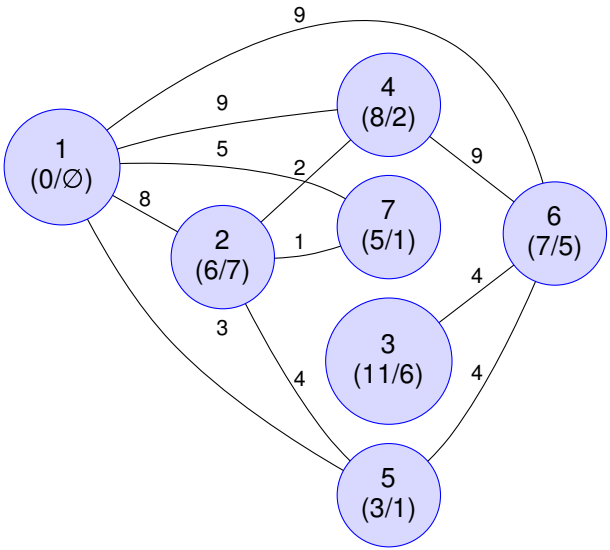


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $5 - 6 - 3$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

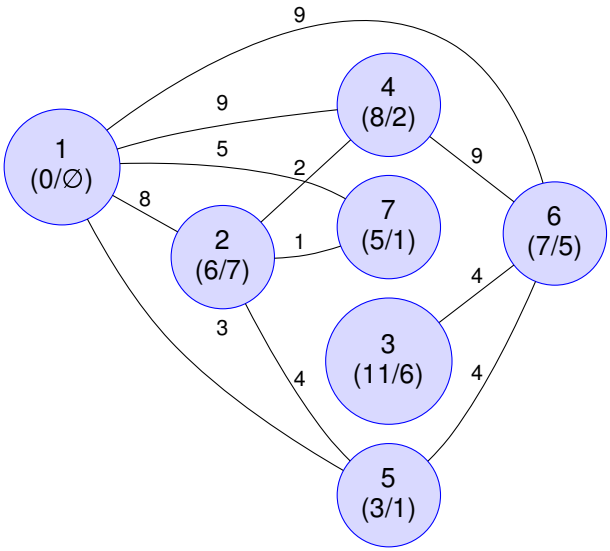


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $1 - 5 - 6 - 3$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

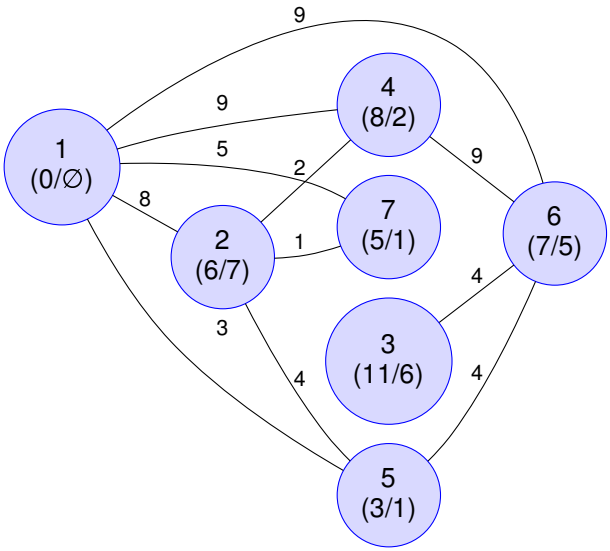


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $1 - 5 - 6 - 3$
- $1 \to 4$: $1 - 7 - 2 - 4$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

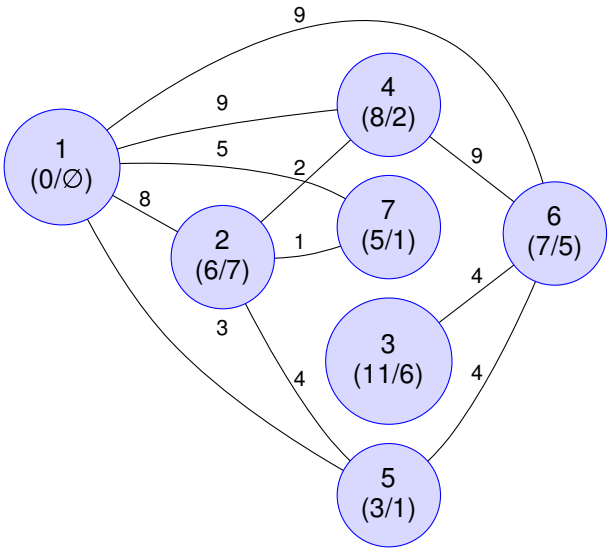


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $1 - 5 - 6 - 3$
- $1 \to 4$: $1 - 7 - 2 - 4$
- $1 \to 5$: $1 - 5$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania

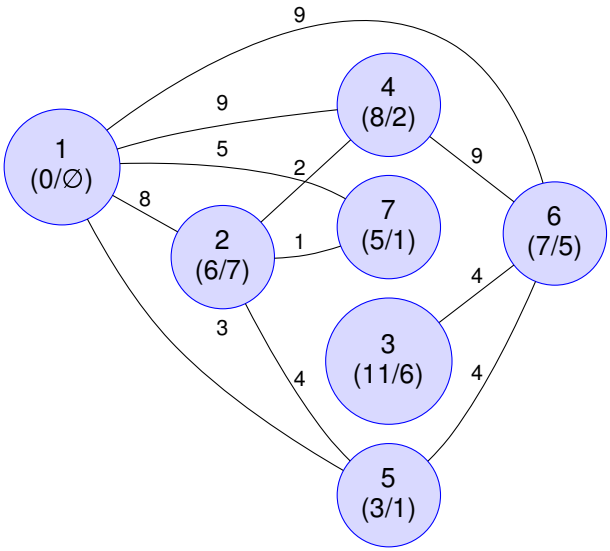


Oznaczenia: $(d_s[u]/p_s[u])$

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $1 - 5 - 6 - 3$
- $1 \to 4$: $1 - 7 - 2 - 4$
- $1 \to 5$: $1 - 5$
- $1 \to 6$: $1 - 5 - 6$

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry – przykład działania



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

Najkrótsze ścieżki

- $1 \to 1$: 1
- $1 \to 2$: $1 - 7 - 2$
- $1 \to 3$: $1 - 5 - 6 - 3$
- $1 \to 4$: $1 - 7 - 2 - 4$
- $1 \to 5$: $1 - 5$
- $1 \to 6$: $1 - 5 - 6$
- $1 \to 7$: $1 - 7$

Podścieżki są najkrótszymi ścieżkami.

Poszukiwanie najkrótszych ścieżek: algorytm Dijkstry

- Algorytm **zachłanny**, ale dowodzi się, że wynik zawsze prawidłowy[a].
- **Złożoność**[b]: (n wyciągnięć u o minimalnym $d_s[u]$) $\times$ (n iteracji) + (k relaksacji) $\Rightarrow O(n^2 + k) = O(n^2)$.
- Zastosowanie tylko dla **nieujemnych wag** (ale może też działać dla grafów skierowanych, o ile wagi nieujemne).

[a] Dowód – patrz: Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., *Wprowadzenie do algorytmów*, Warszawa, Wydawnictwo Naukowo – Techniczne

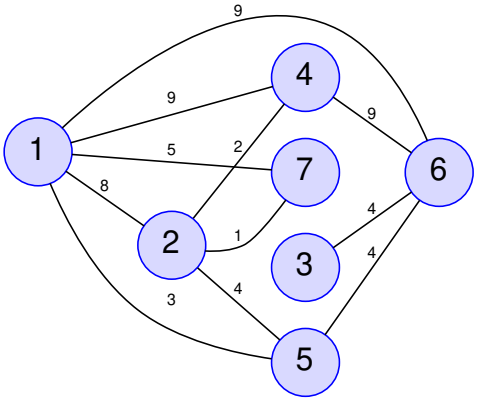
[b] Ale zależy od implementacji wyboru wierzchołka o najmniejszym $d_s[u]$: przy nieoptymalnej może być większa; ale może być mniejsza (kolejka niegotowych wierzchołków przy pomocy kopca binarnego: $O(k \cdot \lg(n))$)

AGH

Zestaw 3, zadanie 3

Wyznaczyć macierz odległości miedzy wszystkimi parami wierzchołków na tym grafie.

Poszukiwanie wszystkich najkrótszych ścieżek



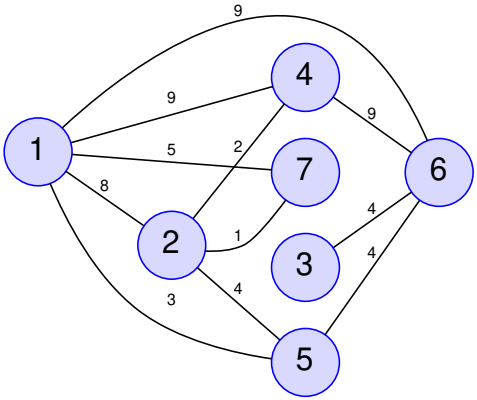
$\texttt{dijkstra}(G, w, s = 1)$

Macierz odległości

- Macierz **kwadratowa** $n \times n$.

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

Poszukiwanie wszystkich najkrótszych ścieżek



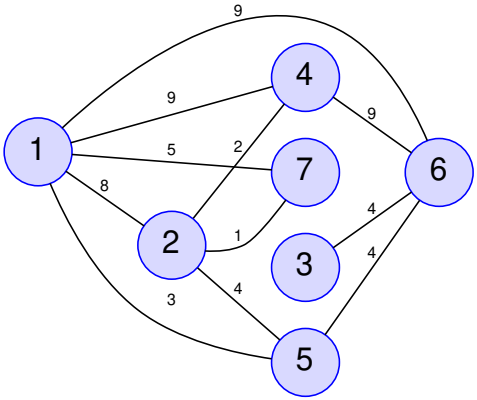
$\texttt{dijkstra}(G, w, s = 2)$

Macierz odległości

- Macierz **kwadratowa** $n \times n$.

	1	2	3	4	5	6	7
1	0	6	11	8	3	7	5
2	6	0	12	2	4	8	1
3							
4							
5							
6							
7							

Poszukiwanie wszystkich najkrótszych ścieżek



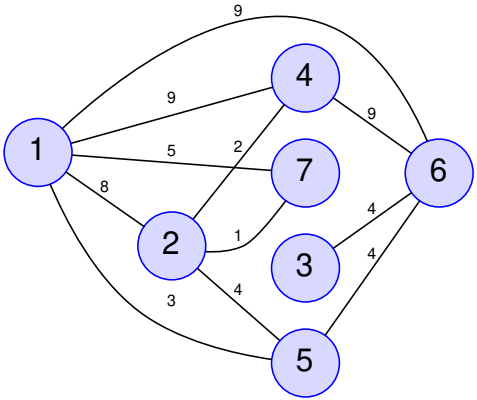
`dijkstra`$(G, w, s = 3)$, itd.

Macierz odległości

- Macierz **kwadratowa** $n \times n$.

	1	2	3	4	5	6	7
1	0	6	11	8	3	7	5
2	6	0	12	2	4	8	1
3	11	12	0	13	8	4	13
4	8	2	13	0	6	9	3
5	3	4	8	6	0	4	5
6	7	8	4	9	4	0	9
7	5	1	13	3	5	9	0

Poszukiwanie wszystkich najkrótszych ścieżek

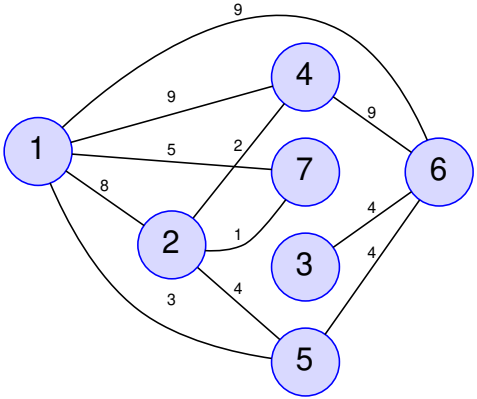


Macierz odległości

- Macierz **kwadratowa** $n \times n$.
- **Diagonala**: $\forall_u D[u][u] = 0$.

	1	2	3	4	5	6	7
1	**0**	6	11	8	3	7	5
2	6	**0**	12	2	4	8	1
3	11	12	**0**	13	8	4	13
4	8	2	13	**0**	6	9	3
5	3	4	8	6	**0**	4	5
6	7	8	4	9	4	**0**	9
7	5	1	13	3	5	9	**0**

Poszukiwanie wszystkich najkrótszych ścieżek



Macierz odległości

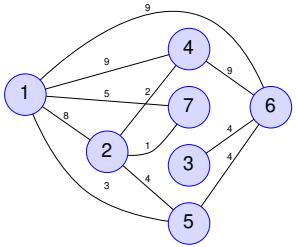
- Macierz **kwadratowa** $n \times n$.
- **Diagonala**: $\forall_u D[u][u] = 0$.
- **Symetryczna** (ponieważ graf nieskierowany).

	1	2	3	4	5	6	7
1	0	6	11	8	3	7	5
2	6	0	12	2	4	8	1
3	11	12	0	13	8	4	13
4	8	2	13	0	6	9	3
5	3	4	8	6	0	4	5
6	7	8	4	9	4	0	9
7	5	1	13	3	5	9	0

Zestaw 3, zadanie 4

Wyznaczyć centrum grafu, to znaczy wierzchołek, którego suma odległości do pozostałych wierzchołków jest minimalna. Wyznaczyć centrum minimax, to znaczy wierzchołek, którego odległość do najdalszego wierzchołka jest minimalna.

Wyznaczanie centrum grafu

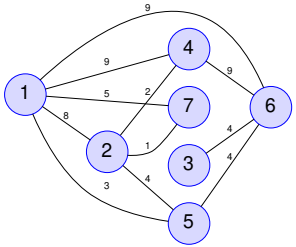


Centrum

Centrum minimax

	1	2	3	4	5	6	7	suma	odległość od najdalszego
1	0	6	11	8	3	7	5		
2	6	0	12	2	4	8	1		
3	11	12	0	13	8	4	13		
4	8	2	13	0	6	9	3		
5	3	4	8	6	0	4	5		
6	7	8	4	9	4	0	9		
7	5	1	13	3	5	9	0		

Wyznaczanie centrum grafu

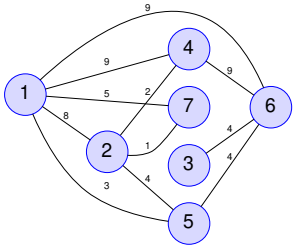


Centrum

Centrum minimax

	1	2	3	4	5	6	7	suma	odległość od najdalszego
1	0	6	11	8	3	7	5	40	
2	6	0	12	2	4	8	1	33	
3	11	12	0	13	8	4	13	61	
4	8	2	13	0	6	9	3	41	
5	3	4	8	6	0	4	5	30	
6	7	8	4	9	4	0	9	41	
7	5	1	13	3	5	9	0	36	

Wyznaczanie centrum grafu



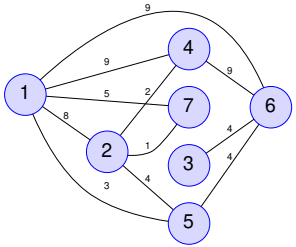
Centrum

5

Centrum minimax

	1	2	3	4	5	6	7	suma	odległość od najdalszego
1	0	6	11	8	3	7	5	40	
2	6	0	12	2	4	8	1	33	
3	11	12	0	13	8	4	13	61	
4	8	2	13	0	6	9	3	41	
5	3	4	8	6	0	4	5	**30**	
6	7	8	4	9	4	0	9	41	
7	5	1	13	3	5	9	0	36	

Wyznaczanie centrum grafu



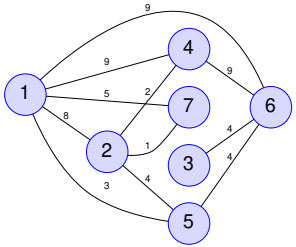
Centrum

5

Centrum minimax

	1	2	3	4	5	6	7	suma	odległość od najdalszego
1	0	6	11	8	3	7	5	40	11
2	6	0	12	2	4	8	1	33	12
3	11	12	0	13	8	4	13	61	13
4	8	2	13	0	6	9	3	41	13
5	3	4	8	6	0	4	5	30	8
6	7	8	4	9	4	0	9	41	9
7	5	1	13	3	5	9	0	36	13

Wyznaczanie centrum grafu



Centrum

5

Centrum minimax

5

	1	2	3	4	5	6	7	suma	odległość od najdalszego
1	0	6	11	8	3	7	5	40	11
2	6	0	12	2	4	8	1	33	12
3	11	12	0	13	8	4	13	61	13
4	8	2	13	0	6	9	3	41	13
5	3	4	8	6	0	4	5	30	**8**
6	7	8	4	9	4	0	9	41	9
7	5	1	13	3	5	9	0	36	13

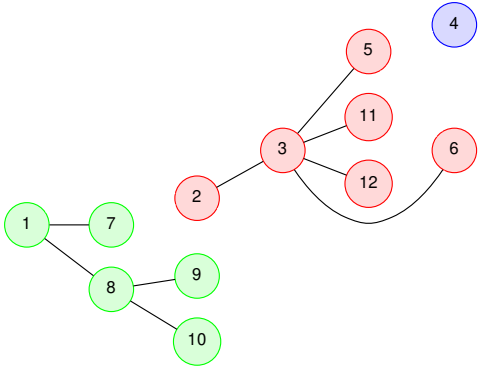
AGH

Zestaw 3, zadanie 5

Wyznaczyć minimalne drzewo rozpinające (algorytm Prima lub Kruskala).

Minimalne drzewo rozpinające

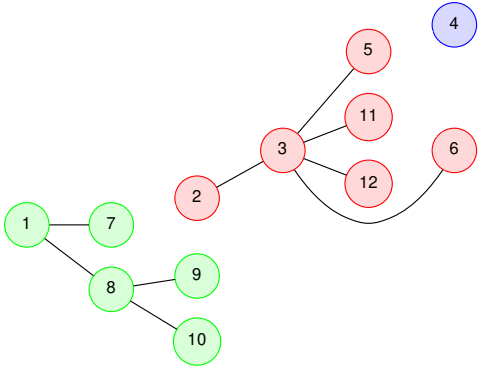
Las – graf bez cykli.

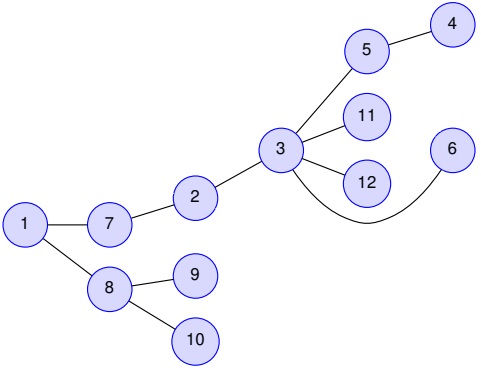


Minimalne drzewo rozpinające

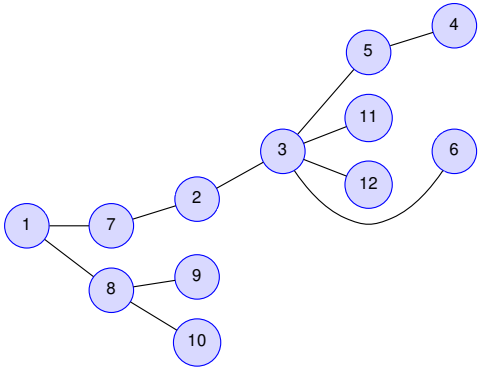
Las – graf bez cykli.

Drzewo – spójny las.
Las składa się z drzew.





Minimalne drzewo rozpinające

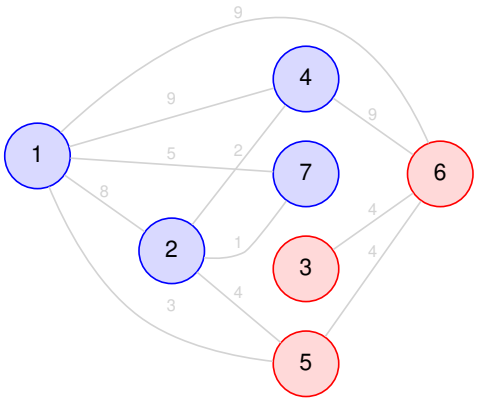


Drzewo

- Każde dwa wierzchołki połączone dokładnie **jedną ścieżką**.
- n wierzchołków $\Rightarrow n-1$ krawędzi.
- Każda krawędź jest **mostem**.
- **Drzewo rozpinające** grafu: złożone ze wszystkich wierzchołków i niektórych krawędzi – tak, by było acykliczne i spójne.
- **Minimalne drzewo rozpinające**[a] grafu: o minimalnej sumie wag.

[a] Zastosowanie: projektowanie układów elektronicznych o najkrótszych przewodach, łączących końcówki układu.

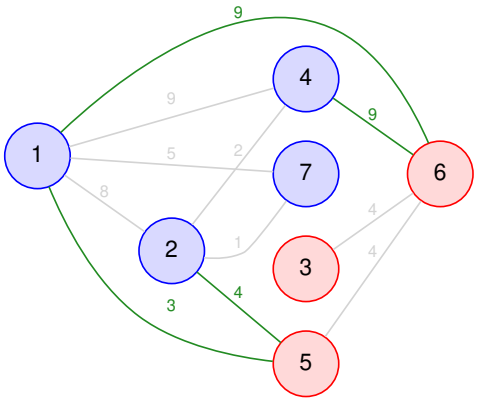
Minimalne drzewo rozpinające: algorytm Prima



Definicje

- **Przekrój**: para podzbiorów wierzchołków T i W taka, że $W = V \setminus T$ (V – zbiór wszystkich wierzchołków).

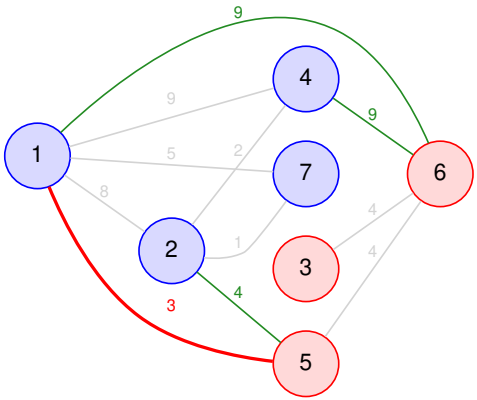
Minimalne drzewo rozpinające: algorytm Prima



Definicje

- **Przekrój**: para podzbiorów wierzchołków T i W taka, że $W = V \setminus T$ (V – zbiór wszystkich wierzchołków).
- **Krawędzie krzyżujące się z przekrojem**: krawędzie łączące wierzchołki należące do T z należącymi do W.

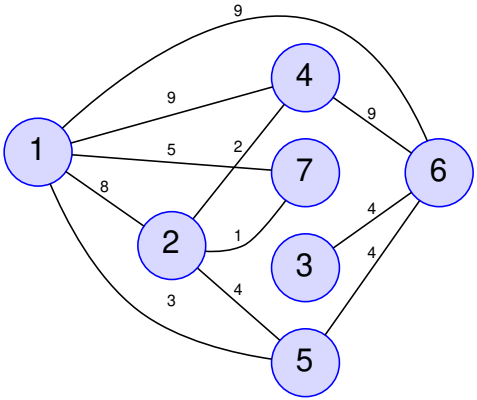
Minimalne drzewo rozpinające: algorytm Prima



Definicje

- **Przekrój**: para podzbiorów wierzchołków T i W taka, że $W = V \setminus T$ (V – zbiór wszystkich wierzchołków).
- **Krawędzie krzyżujące się z przekrojem**: krawędzie łączące wierzchołki należące do T z należącymi do W.
- **Krawędź lekka**: krawędź krzyżująca się z przekrojem, która ma najmniejszą wagę.

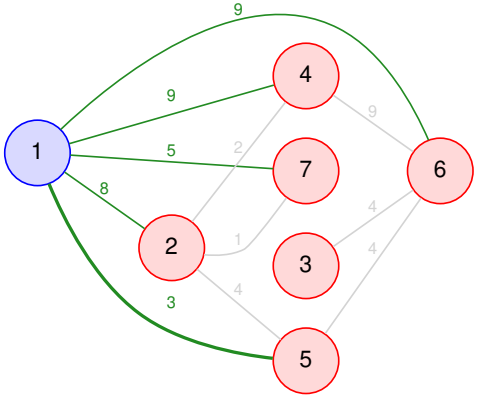
Minimalne drzewo rozpinające: algorytm Prima



Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

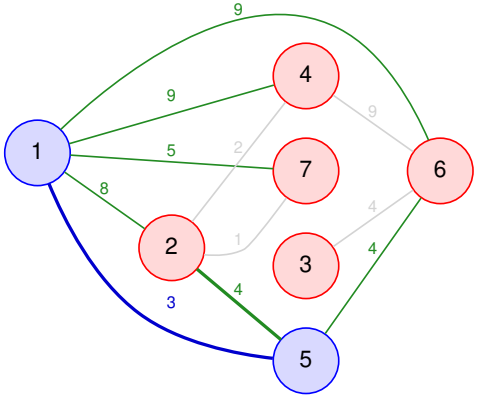


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - **Do zbioru T dodajemy krawędź lekką.**
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

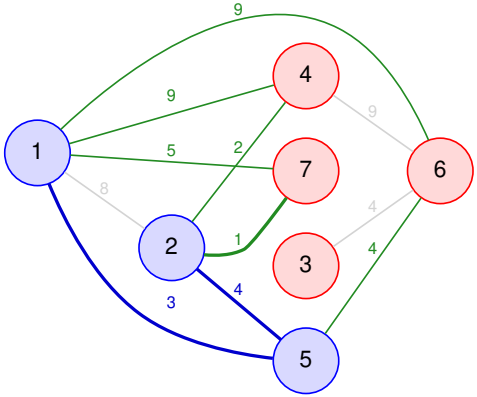


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

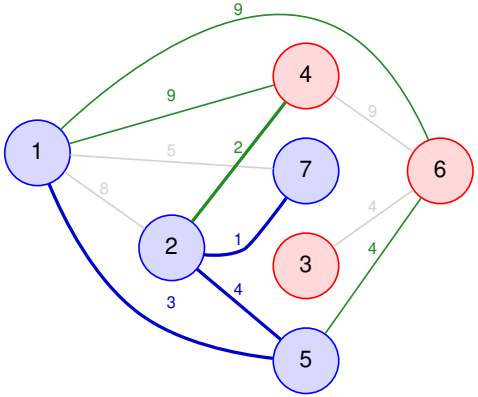


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

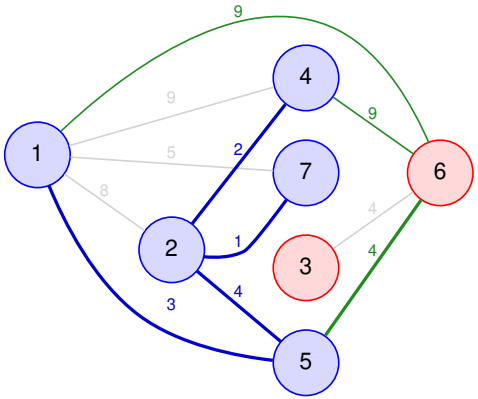


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

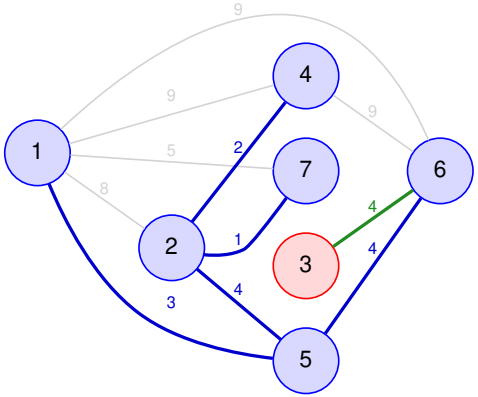


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Prima

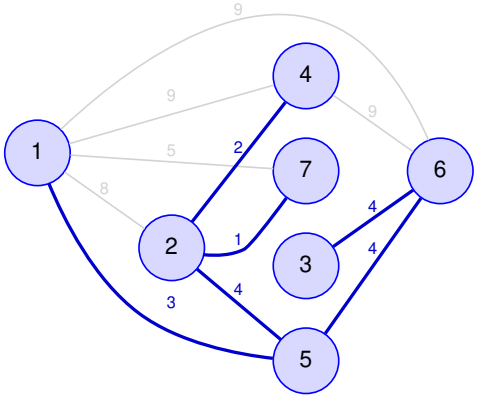


Podzbiór T: niebieskie wierzchołki i krawędzie (drzewo).
Podzbiór W: czerwone wierzchołki (pozostałe).
Krawędzie krzyżujące się z przekrojem: zielone.
Krawędź lekka: **pogrubiona zielona**.

Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.

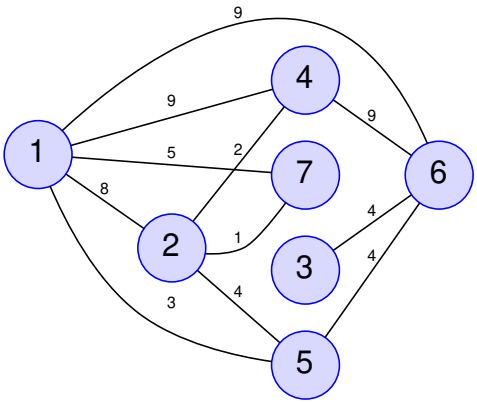
Minimalne drzewo rozpinające: algorytm Prima



Algorytm Prima

1. Dowolny startowy wierzchołek: dodajemy do drzewa T.
2. W każdym kroku analizujemy tylko krawędzie **krzyżujące się z przekrojem**.
   - Do zbioru T dodajemy **krawędź lekką**.
3. **Gdy w T są wszystkie wierzchołki $\Rightarrow$ koniec.**

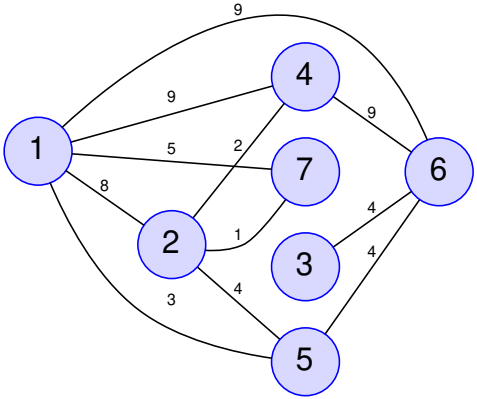
Minimalne drzewo rozpinające: algorytm Kruskala



Algorytm Kruskala

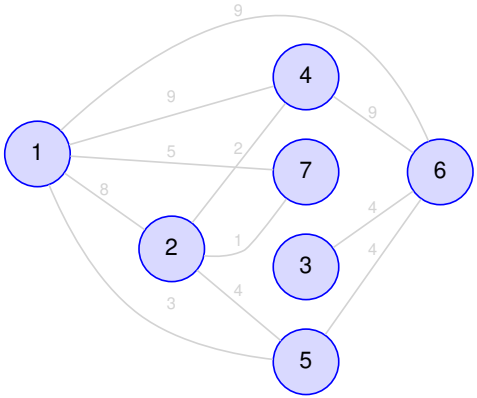
1. Do podzbioru T dodajemy wszystkie wierzchołki.
2. Sortujemy krawędzie według niemalejących wag.
3. W zadanej kolejności:
   - Jeśli krawędź łączy różne drzewa ($\equiv$ nie powoduje powstania cyklu) $\Rightarrow$ dodajemy ją do drzewa T.
4. Gdy w T jest $n-1$ krawędzi $\Rightarrow$ koniec.

Minimalne drzewo rozpinające: algorytm Kruskala



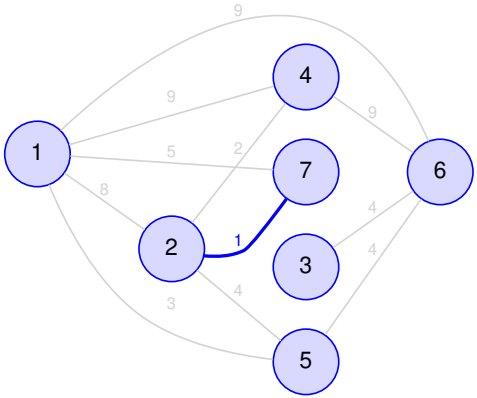
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



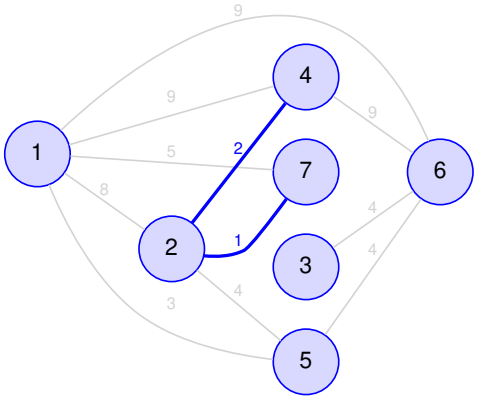
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



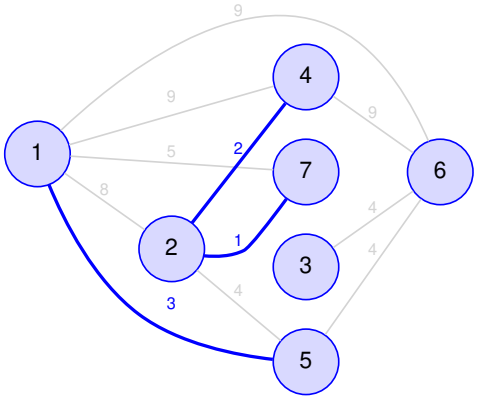
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



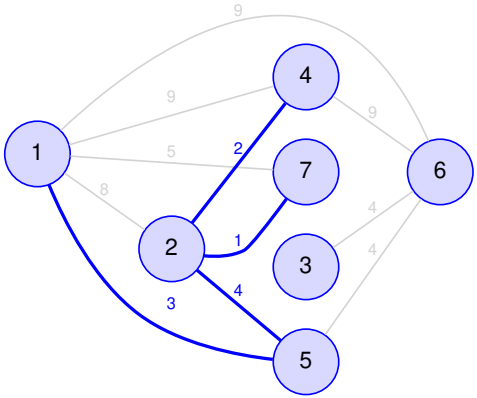
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



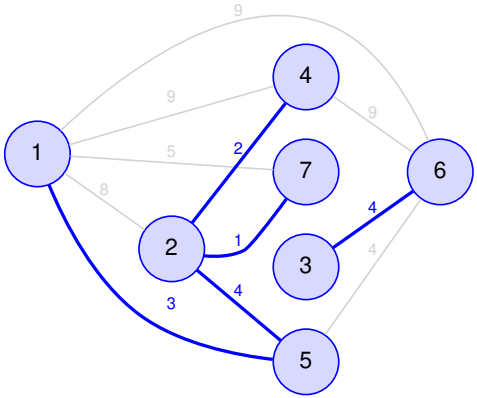
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



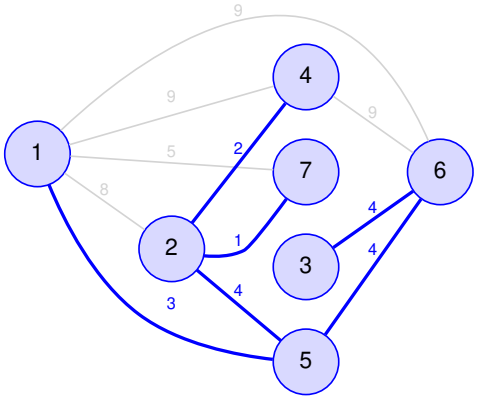
Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala



Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Minimalne drzewo rozpinające: algorytm Kruskala

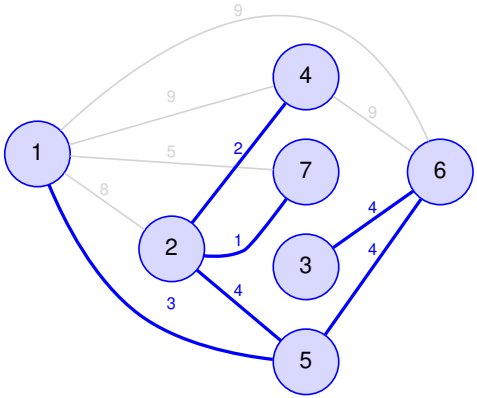


Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
(1,7)	5
(1,2)	8
(1,4)	9
(1,6)	9
(4,6)	9

Uwaga

Nie zawsze wszystkie pierwsze krawędzie będą należały do drzewa – trzeba omijać te, które spowodowałyby powstanie cyklu.

Minimalne drzewo rozpinające: algorytm Kruskala



Krawędź (u, v)	Waga krawędzi $w(u, v)$
(2,7)	1
(2,4)	2
(1,5)	3
(2,5)	4
(3,6)	4
(5,6)	4
~~(1,7)~~	5
~~(1,2)~~	8
~~(1,4)~~	9
~~(1,6)~~	9
~~(4,6)~~	9

Uwaga

Nie zawsze wszystkie pierwsze krawędzie będą należały do drzewa – trzeba omijać te, które spowodowałyby powstanie cyklu.

Algorytm Prima i algorytm Kruskala: porównanie

Podobieństwa

- Start: zbiór T bez krawędzi, do którego dodajemy krawędzie bezpieczne.
- Niezmiennik algorytmu: T w każdym kroku jest podzbiorem minimalnego drzewa rozpinającego.
- Złożoność $O(k \lg(n))$.
- Algorytmy zachłanne (ale wynik zawsze prawidłowy).

Różnice

- Sposób wyszukiwania krawędzi bezpiecznych.