Drumuri minime de sursă unică în grafuri aciclice

Drumuri minime de sursă unică în grafuri aciclice

Ipoteze:

- Graful **nu** conține circuite
- Arcele pot avea **și cost negativ**

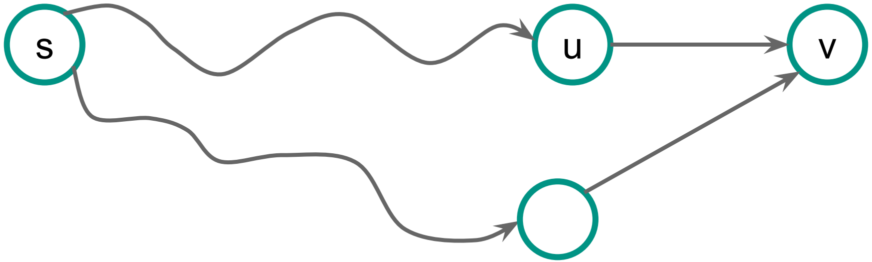
Drumuri minime de sursă unică în grafuri aciclice

Amintim:

- Când considerăm un vârf v, pentru a calcula d(s, v) ar fi util să știm deja δ(s, u), pentru orice u cu uv$\in$E

- **atunci, putem calcula distanțele după relația**

$$\delta(s, v) = \min \{ \delta(s, u) + w(u, v) \mid uv \in E \}$$



Drumuri minime de sursă unică în grafuri aciclice

Amintim:

- Când considerăm un vârf v, pentru a calcula d(s, v) ar fi util să știm deja δ(s, u), pentru orice u cu uv∈E

  ⇒

  ar fi utilă o ordonare a vârfurilor, astfel încât, dacă uv∈E, atunci u se află înaintea lui v

Drumuri minime de sursă unică în grafuri aciclice

Amintim:

- Când considerăm un vârf v, pentru a calcula d(s, v) ar fi util să știm deja δ(s, u), pentru orice u cu uv∈E

  ⇒

  ar fi utilă o ordonare a vârfurilor, astfel încât, dacă uv∈E, atunci u se află înaintea lui v

O astfel de ordonare <u>nu există</u> dacă graful conține circuite

Drumuri minime de sursă unică în grafuri aciclice

Amintim:

- Când considerăm un vârf v, pentru a calcula d(s, v) ar fi util să știm deja δ(s, u), pentru orice u cu uv∈E

  ⇒

  ar fi utilă o ordonare a vârfurilor, astfel încât, dacă uv∈E, atunci u se află înaintea lui v

O astfel de ordonare <u>există</u> dacă graful <u>nu</u> conține circuite = sortarea topologică

Pseudocod

Drumuri minime de sursă unică în grafuri aciclice

- **Considerăm vârfurile în ordinea dată de sortarea topologică**
- **Pentru fiecare vârf u, relaxăm arcele uv către vecinii săi (pentru a găsi drumuri noi către aceștia)**

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
// este suficient să păstrăm vârfurile din sortare începând cu s
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
// este suficient să păstrăm vârfurile din sortare începând cu s
SortTop = sortare_topologică(G, s)

pentru fiecare u∈SortTop execută
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
// este suficient să păstrăm vârfurile din sortare începând cu s
SortTop = sortare_topologică(G, s)

pentru fiecare u∈SortTop execută
    pentru fiecare uv∈E execută
```

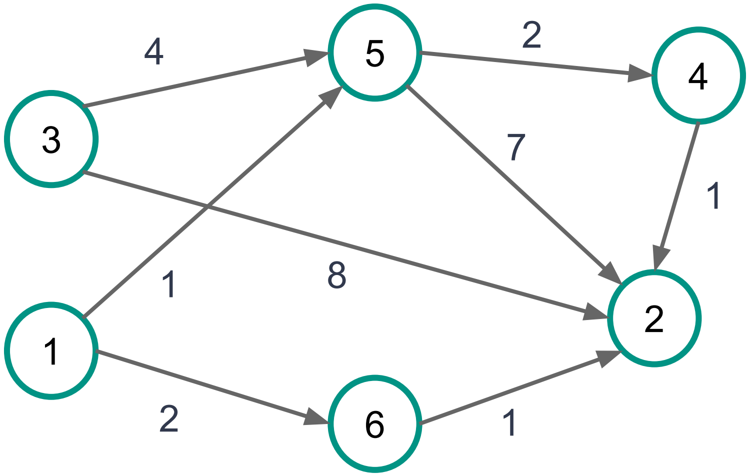
Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
// este suficient să păstrăm vârfurile din sortare începând cu s
SortTop = sortare_topologică(G, s)

pentru fiecare u∈SortTop execută
    pentru fiecare uv∈E execută
        dacă d[u]+w(u,v) < d[v] atunci// relaxăm uv
            d[v] = d[u]+w(u,v)
            tata[v] = u
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
// este suficient să păstrăm vârfurile din sortare începând cu s
SortTop = sortare_topologică(G, s)

pentru fiecare u∈SortTop execută
    pentru fiecare uv∈E execută
        dacă d[u]+w(u,v) < d[v] atunci// relaxăm uv
            d[v] = d[u]+w(u,v)
            tata[v] = u

scrie d, tata
```

Exemplu

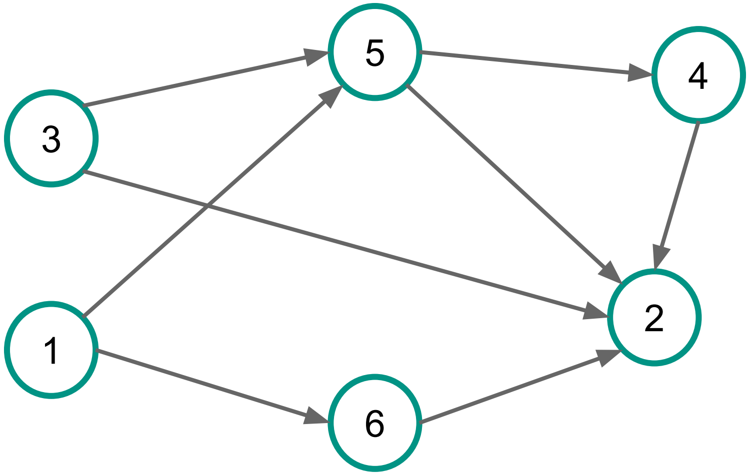


Exemplu

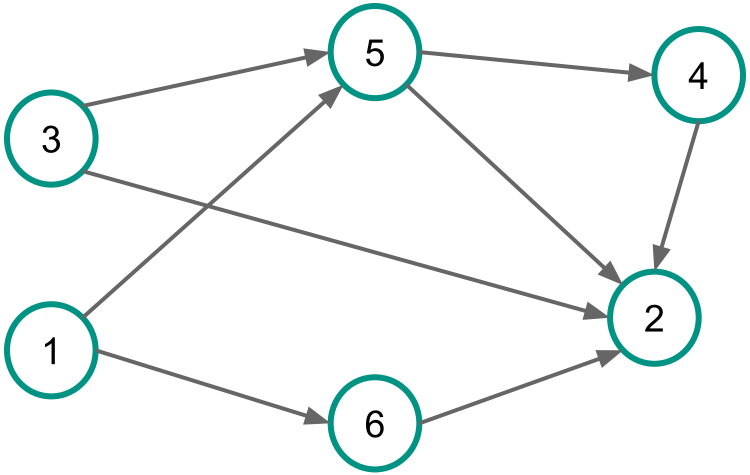
- **<u>Etapa 1</u> - determinăm o ordonare topologică a vârfurilor**
- **Amintim algoritm**

```
SortTop ← ∅
coada C ← ∅
adaugă în C toate vârfurile v cu d⁻[v] = 0
cât timp C ≠ ∅ execută
    i ← extrage(C)
    adaugă i în SortTop
    pentru ij ∈ E execută
        d⁻[j]--
        dacă d⁻[j] = 0 atunci
            adaugă(j, C)

returnează SortTop
```

Sortare topologică – Exemplu

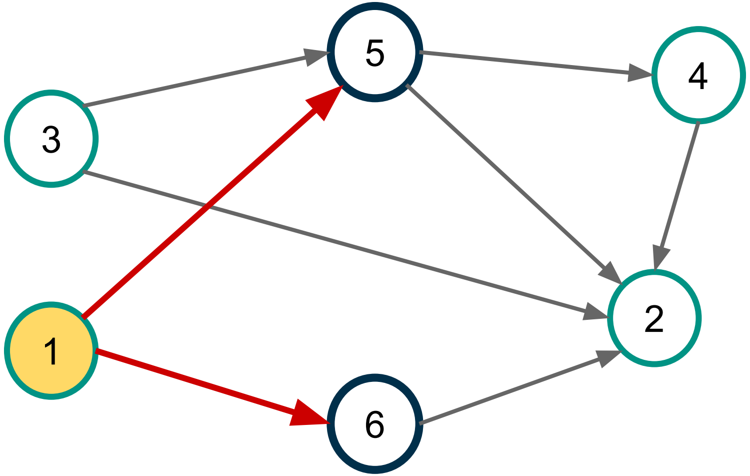


Sortare topologică – Exemplu



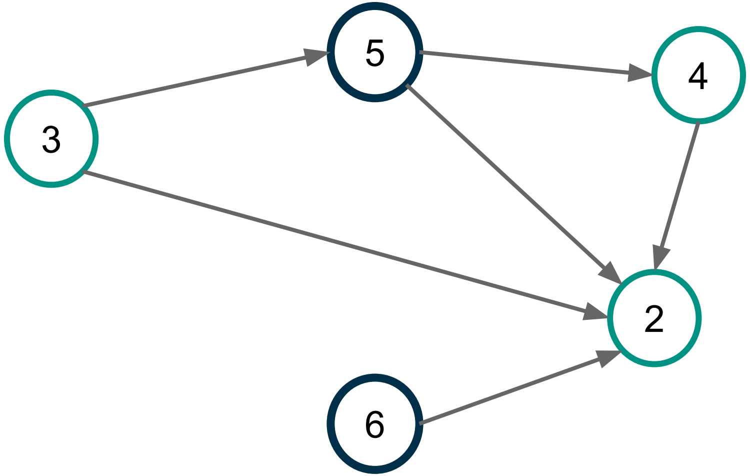
C: 1 3

Sortare topologică – Exemplu



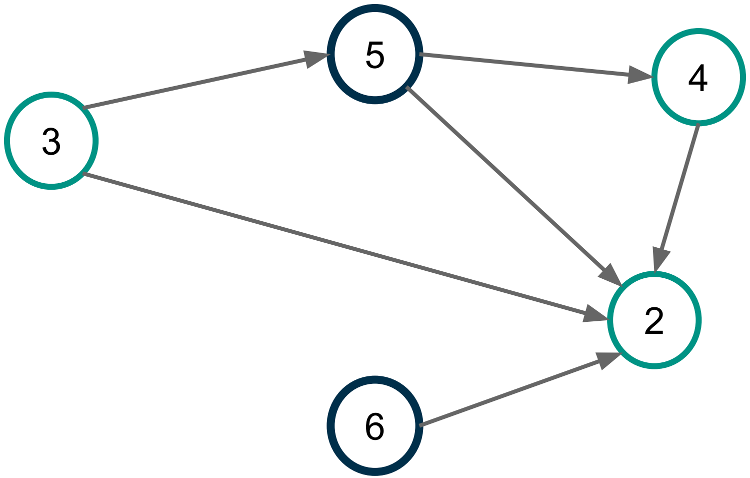
C: 1 3

Sortare topologică – Exemplu



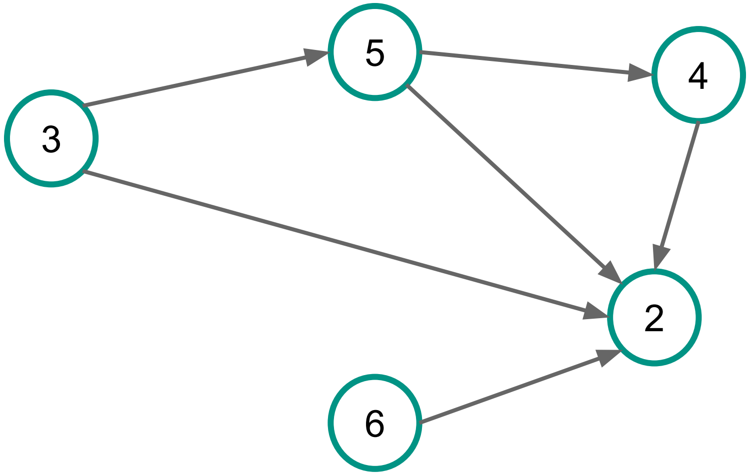
C: **1** 3

Sortare topologică – Exemplu



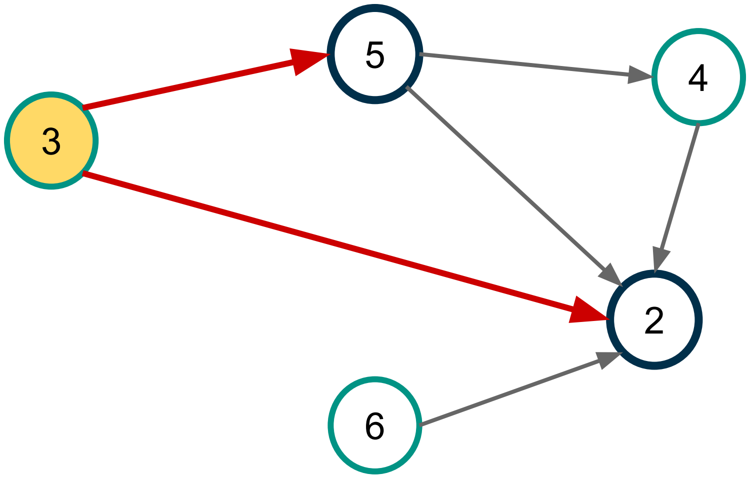
C: 1 3 6

Sortare topologică – Exemplu



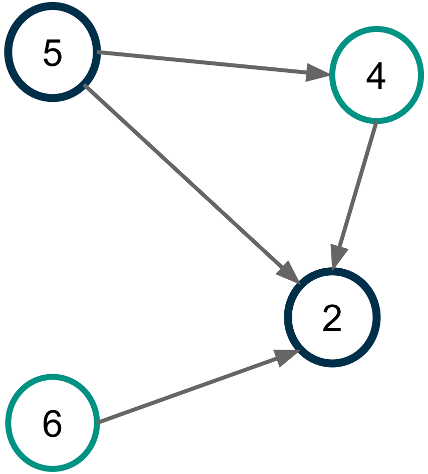
C: 1 3 6

Sortare topologică – Exemplu



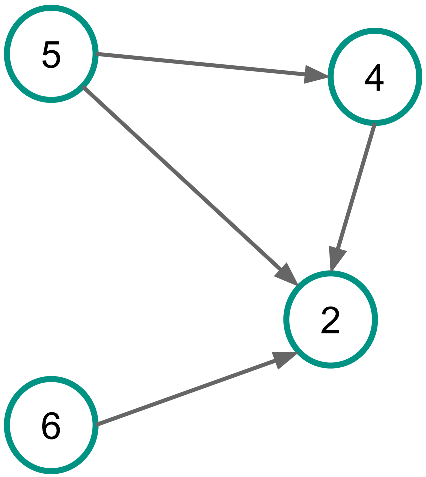
C: **1** **3** 6

Sortare topologică – Exemplu



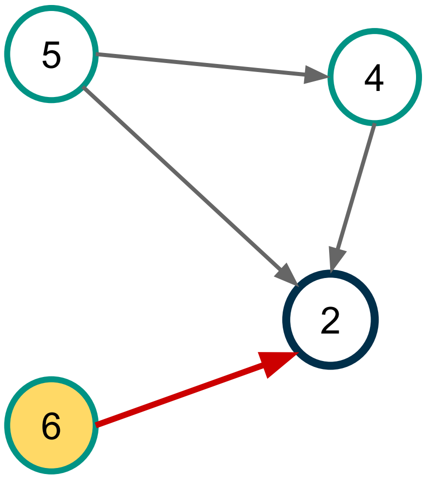
C: **1** **3** 6

Sortare topologică – Exemplu



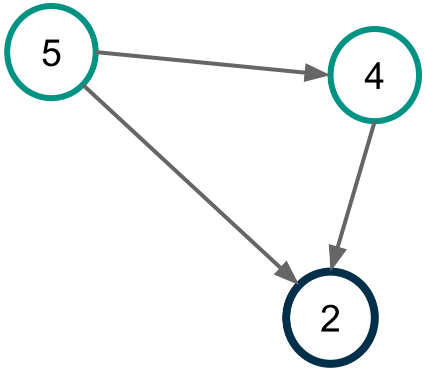
C: 1 3 6 5

Sortare topologică – Exemplu



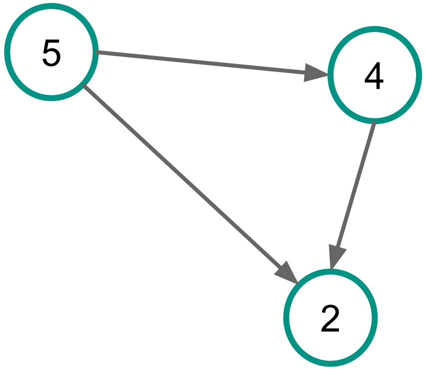
C: **1 3 6** 5

Sortare topologică – Exemplu



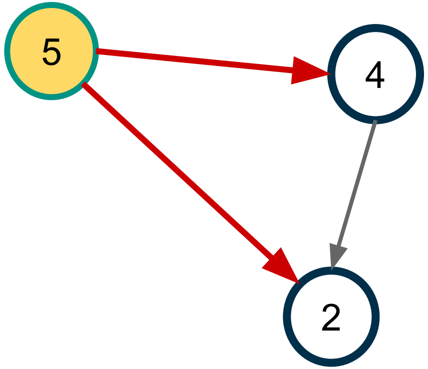
C: **1 3 6** 5

Sortare topologică – Exemplu



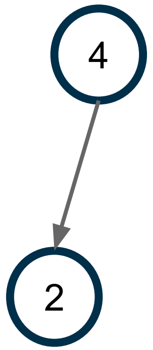
C: 1 3 6 5

Sortare topologică – Exemplu



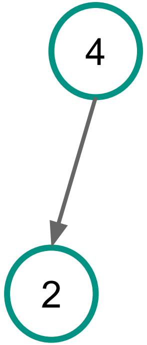
C: **1 3 6 5**

Sortare topologică – Exemplu



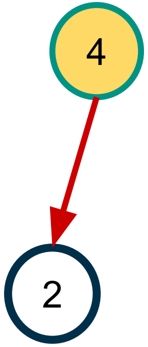
C: **1 3 6 5**

Sortare topologică – Exemplu



C: 1 3 6 5 4

Sortare topologică – Exemplu



C: 1 3 6 5 4

Sortare topologică – Exemplu

C: **1 3 6 5 4**

Sortare topologică – Exemplu

C: **1 3 6 5 4** 2

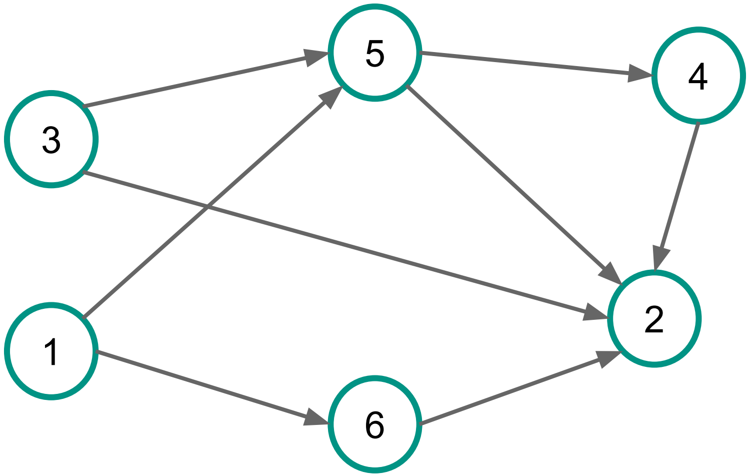
Sortare topologică – Exemplu

C: **1 3 6 5 4 2**

Sortare topologică – Exemplu

C: 1 3 6 5 4 2

Sortare topologică – Exemplu



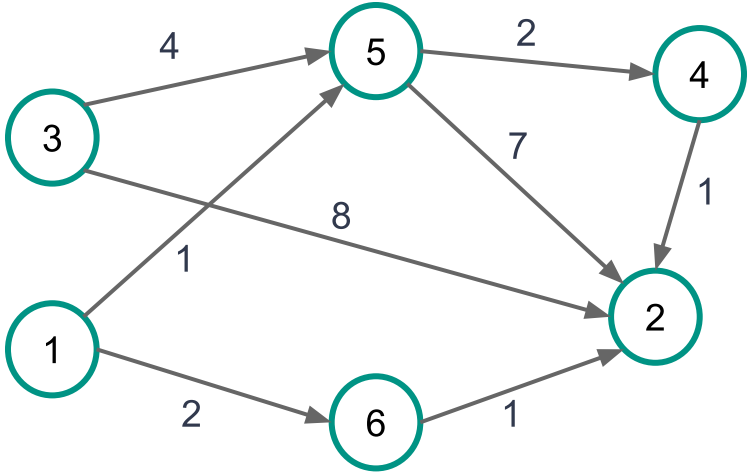
SORTARE TOPOLOGICĂ: 1 3 6 5 4 2

Sortare topologică – Algoritm

```
coada C ← ∅
adaugă în C toate vârfurile v cu d⁻[v]=0

cât timp C ≠ ∅ execută
    i ← extrage(C)
    adaugă i în sortare
    pentru ij ∈ E execută
        d⁻[j]--
        dacă d⁻[j] = 0 atunci
            adaugă(j, C)

return C
```

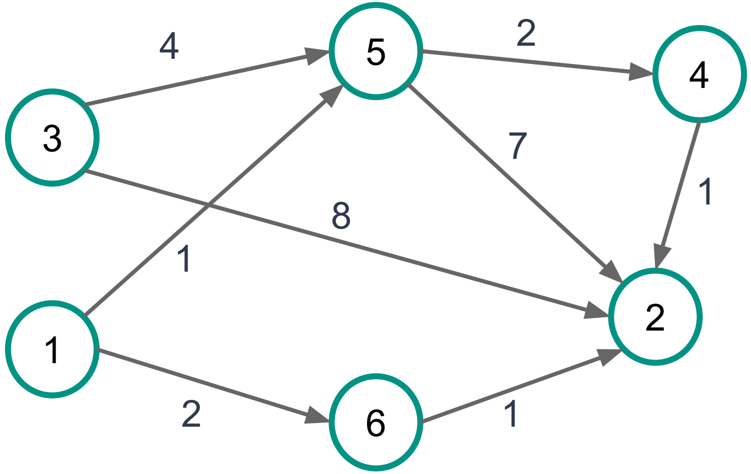
Exemplu

- Etapa 2 - parcurgem vârfurile în ordinea dată de sortarea topologică și relaxăm, pentru fiecare vârf, arcele care ies din acesta



Sortare topologică

1, 3, 6, 5, 4, 2



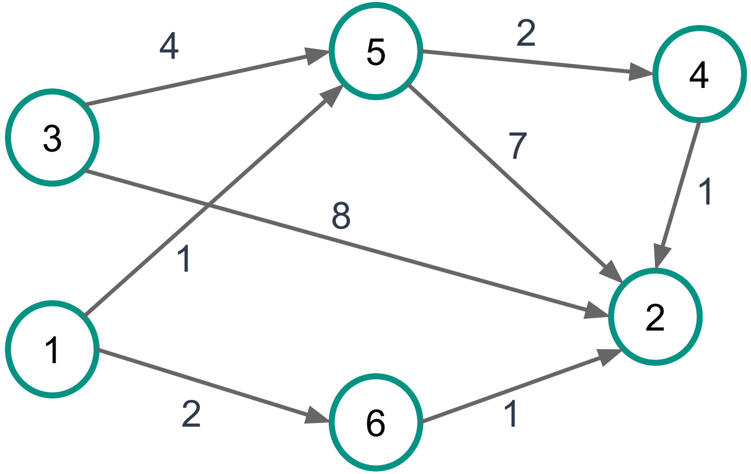
Sortare topologică

1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

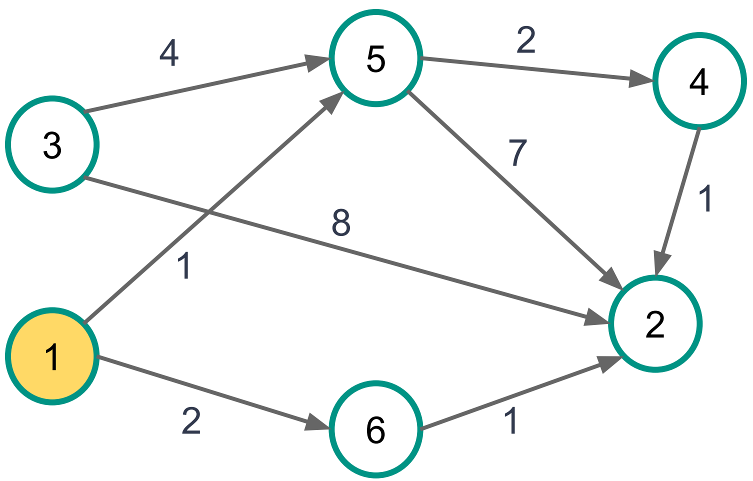


Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	**[∞/0,**	**∞/0,**	**0/0,**	**∞/0,**	**∞/0,**	**∞/0]**



Sortare topologică

1, 3, 6, 5, 4, 2

s = 3 - vârf de start

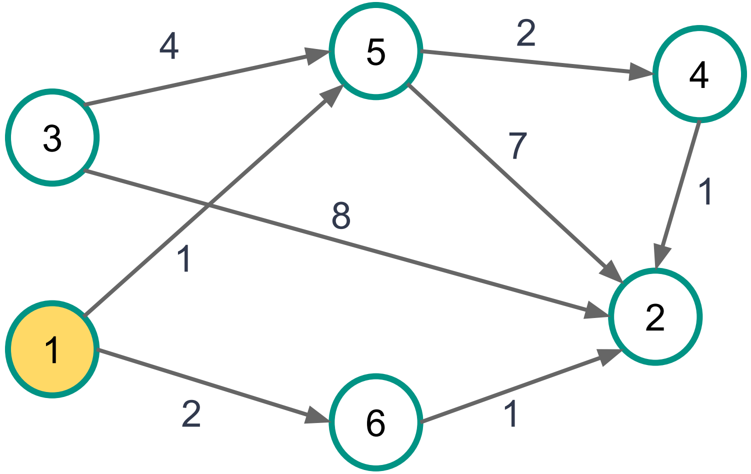
Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]

u = 1

d[v] = min { d[v], d[u]+w(u,v) }



Sortare topologică
1, 3, 6, 5, 4, 2

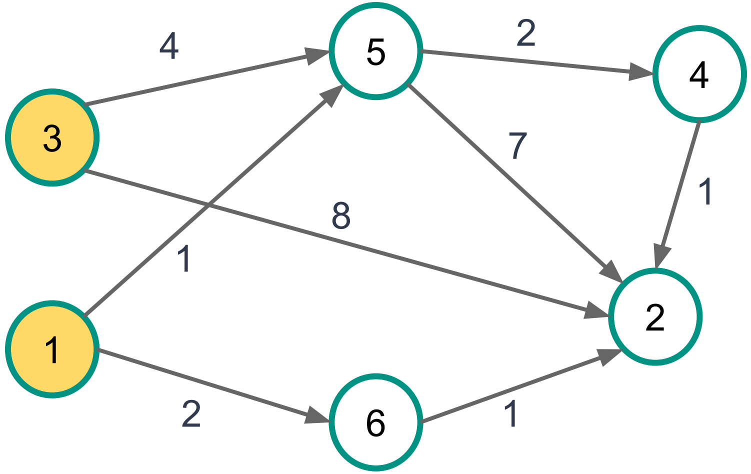
s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]

1 nu este accesibil din s - putem să nu îl considerăm
(să ignorăm vârfurile din ordonarea topologică aflate înaintea lui s)

d[v] = min { d[v], d[u]+w(u,v) }



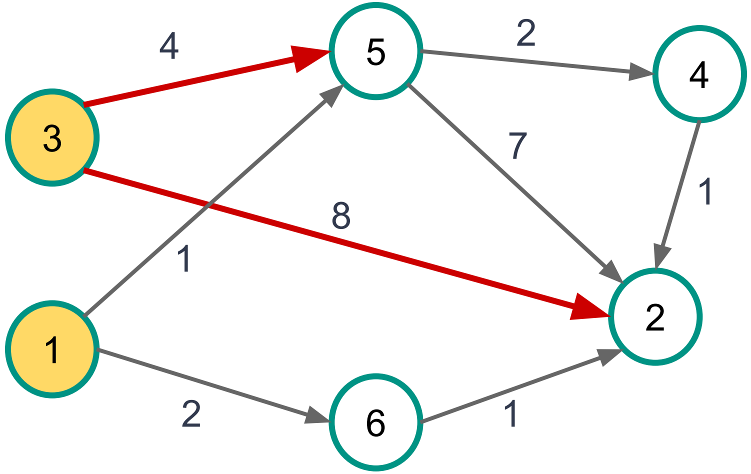
Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3						

d[v] = min { d[v], d[u]+w(u,v) }



Sortare topologică

1, 3, 6, 5, 4, 2

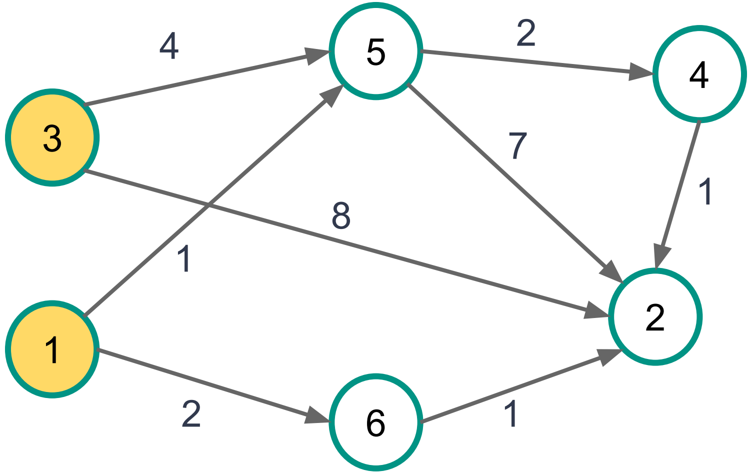
s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3						

$d[v] = \min \{ d[v], d[u]+w(u,v) \}$



Sortare topologică

1, 3, 6, 5, 4, 2

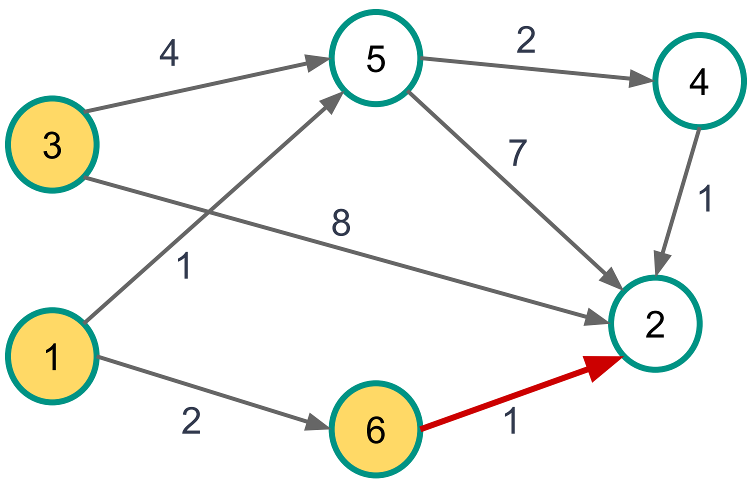
s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]

d[v] = min { d[v], d[u]+w(u,v) }



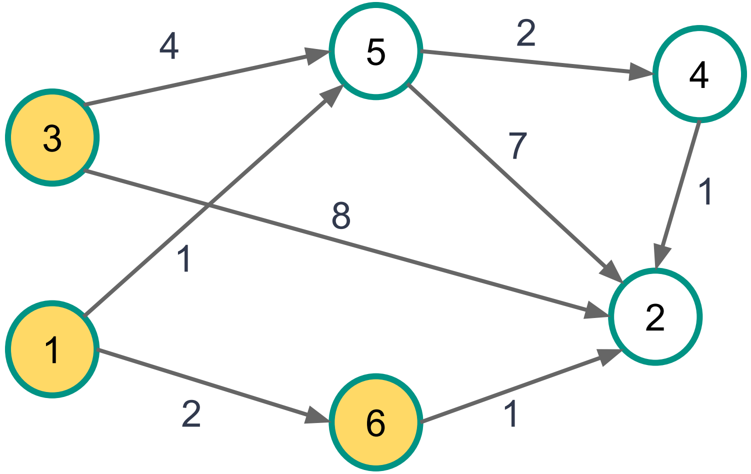
Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6						

d[v] = min { d[v], d[u]+w(u,v) }



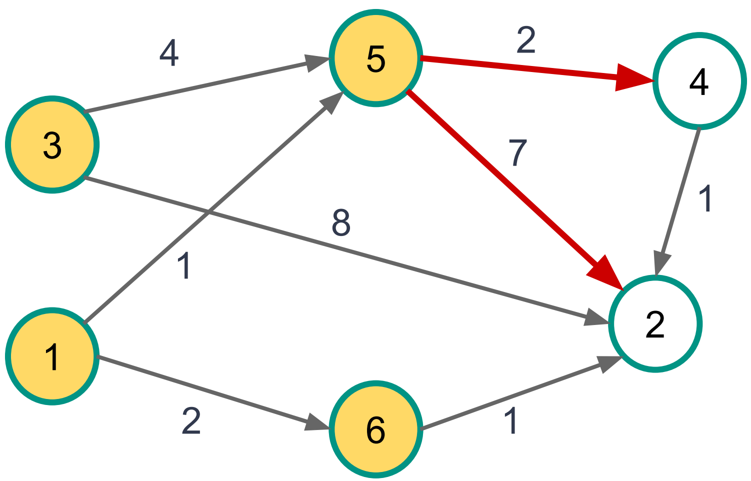
Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]

$d[v] = \min \{ d[v], d[u]+w(u,v) \}$



Sortare topologică

1, 3, 6, 5, 4, 2

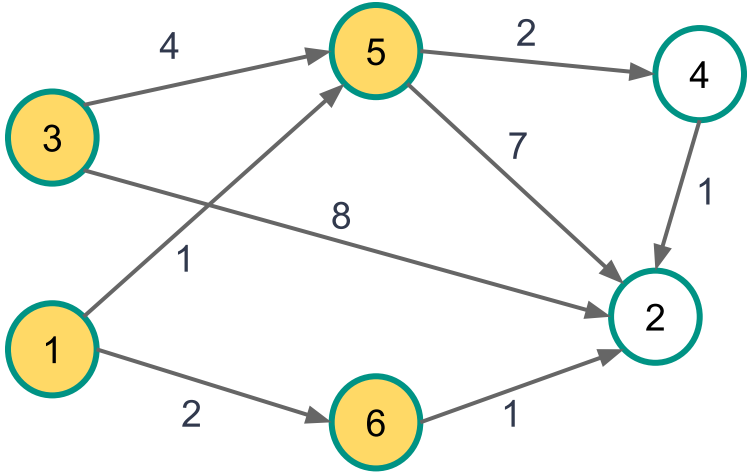
s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5						

d[v] = min { d[v], d[u]+w(u,v) }



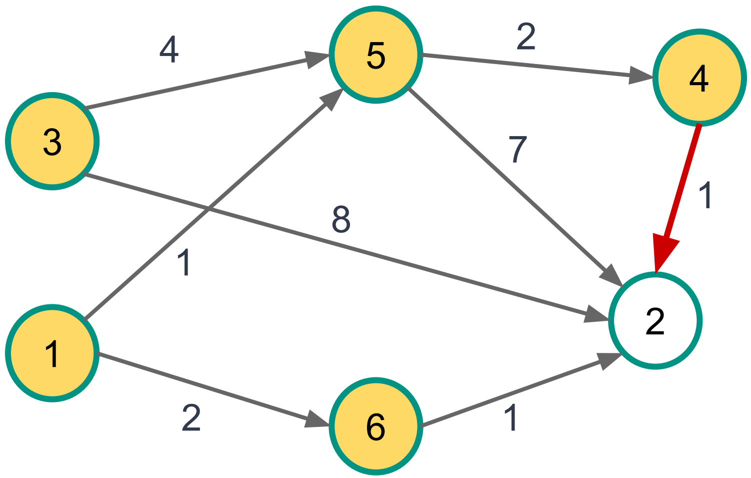
Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5	[∞/0,	8/3,	0/0,	6/5,	4/3,	∞/0]

d[v] = min { d[v], d[u]+w(u,v) }



Sortare topologică

1, 3, 6, 5, 4, 2

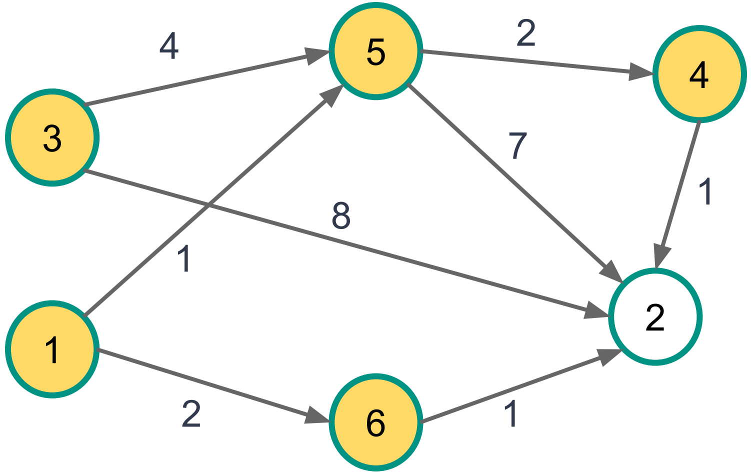
s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5	[∞/0,	8/3,	0/0,	6/5,	4/3,	∞/0]
u = 4						

$d[v] = \min \{ d[v], d[u]+w(u,v) \}$



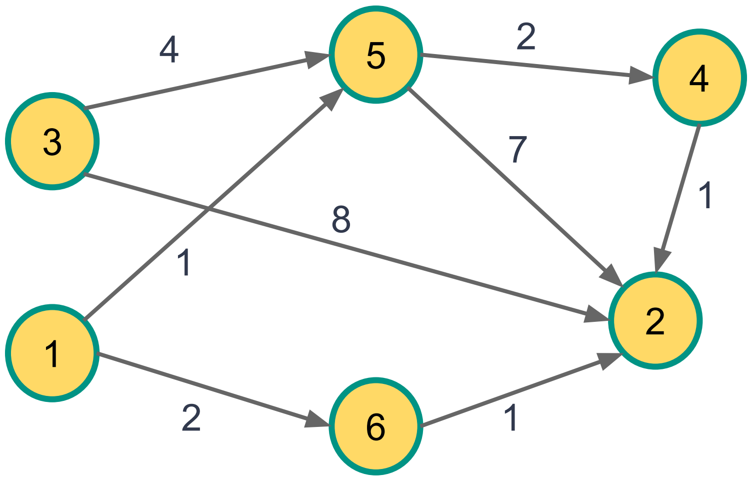
Sortare topologică
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe
1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5	[∞/0,	8/3,	0/0,	6/5,	4/3,	∞/0]
u = 4	[∞/0,	7/4,	0/0,	6/5,	4/3,	∞/0]

$d[v] = \min \{ d[v], d[u]+w(u,v) \}$



Sortare topologică

1, 3, 6, 5, 4, 2

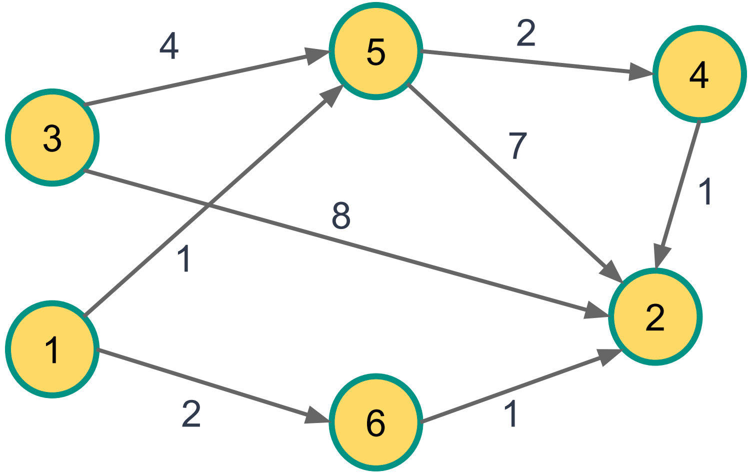
s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5	[∞/0,	8/3,	0/0,	6/5,	4/3,	∞/0]
u = 4	[∞/0,	7/4,	0/0,	6/5,	4/3,	∞/0]
u = 2						

$d[v] = \min \{ d[v], d[u]+w(u,v) \}$



Sortare topologică

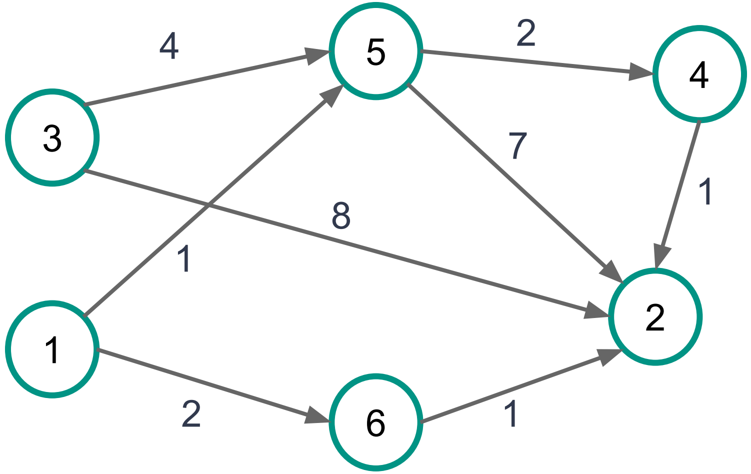
1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
d/tata	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 1	[∞/0,	∞/0,	0/0,	∞/0,	∞/0,	∞/0]
u = 3	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 6	[∞/0,	8/3,	0/0,	∞/0,	4/3,	∞/0]
u = 5	[∞/0,	8/3,	0/0,	6/5,	4/3,	∞/0]
u = 4	[∞/0,	7/4,	0/0,	6/5,	4/3,	∞/0]
u = 2	[∞/0,	7/4,	0/0,	6/5,	4/3,	∞/0]



Sortare topologică

1, 3, 6, 5, 4, 2

s = 3 - vârf de start

Ordine de calcul distanțe

1, 3, 6, 5, 4, 2

	1	2	3	4	5	6
Soluție	[∞/0,	7/4,	0/0,	6/5,	4/3,	∞/0]

Un drum minim de la 3 la 2?

Drumuri minime de sursă unică în grafuri aciclice

Observație

- Este suficient să considerăm, în ordonarea topologică, doar vârfurile accesibile din s
- **În exemplu** - fără 1 și 6

Complexitate

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start
void df(int i) {
    viz[i] = 1;
    for ij ∈ E
        if (viz[j] == 0)
            df(j);
    // i este finalizat
    push(S, i)
}
for (i=1; i<=n; i++)
    if (viz[i] == 0)
        df(i);
while (not S.empty()) {
    u = S.pop();
    adaugă u în sortare
}
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

// inițializăm distanțe - ca la Dijkstra
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0
d[s] = 0

// determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologică(G)

pentru fiecare u∈SortTop execută
    pentru fiecare uv∈E execută
        dacă d[u]+w(u,v) < d[v] atunci// relaxăm uv
            d[v] = d[u]+w(u,v)
            tata[v] = u

scrie d, tata
```

Drumuri minime de sursă unică în grafuri aciclice

Complexitate

- **Inițializare** → O(?)
- **Sortare topologică** → O(?)
- **m * relaxare uv** → O(?)

Drumuri minime de sursă unică în grafuri aciclice

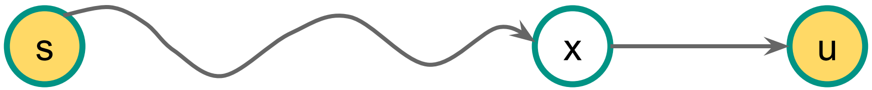
Complexitate

Inițializare	→	O(n)
Sortare topologică	→	O(m + n)
m * relaxare uv	→	O(m)
		O(m + n)

Corectitudine

Drumuri minime de sursă unică în grafuri aciclice

Algoritmul funcționează corect și dacă există arce cu cost negativ



$$\delta(s, u) = \min \{ \delta(s, x) + w(x, u) \mid xu \in E \}$$

Când algoritmul ajunge la vârful u, avem:

$$d[u] = \min \{ d[x] + w(x, u) \mid xu \in E \}$$

relaxare arce xu