Drumuri minime de sursă unică în grafuri aciclice (fără circuite)

Drumuri minime de sursă unică în grafuri aciclice

- **Ipoteze:**
  - **Graful nu conţine circuite**
  - **Arcele pot avea şi cost negativ**

Drumuri minime de sursă unică în grafuri aciclice

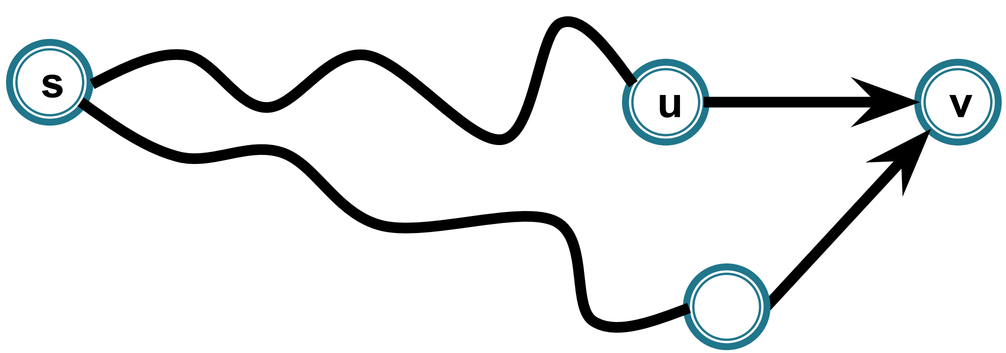
- **Amintim:**

Când considerăm un vârf v, pentru a calcula d(s,v) ar fi util

să ştim deja δ(s,u) pentru orice u cu uv∈E

- **atunci putem calcula distanţele după relaţia**

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



Drumuri minime de sursă unică în grafuri aciclice

- **Amintim:**

  Când considerăm un vârf v, pentru a calcula d(s,v) ar fi util

  să ştim 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**

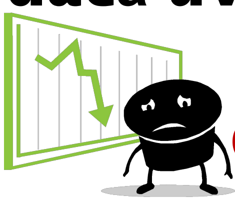
Drumuri minime de sursă unică în grafuri aciclice

- **Amintim:**

  Când considerăm un vârf v, pentru a calcula d(s,v) ar fi util

  să ştim 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</u> <u>există</u> dacă graful conţine circuite

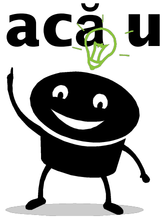
Drumuri minime de sursă unică în grafuri aciclice

- **Amintim:**

Când considerăm un vârf v, pentru a calcula d(s,v) ar fi util

să ştim 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> conţine 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

//initializam distante - ca la Dijkstra
pentru fiecare u∈V executa
        d[u] = ∞; tata[u]=0
d[s] = 0

//determinăm o sortare topologică a vârfurilor
//este suficient sa pastrăm vârfurile din sortare începând cu s
SortTop = sortare_topologica(G,s)

pentru fiecare u ∈ SortTop
        pentru fiecare uv∈E executa
            daca d[u]+w(u,v)<d[v] atunci  //relaxam uv
                  d[v] = d[u]+w(u,v)
               tata[v] = u
```

Drumuri minime de sursă unică în grafuri aciclice

```
s – vârful de start


//initializam distante – ca la Dijkstra
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

//initializam distante - ca la Dijkstra
pentru fiecare u∈V executa
        d[u] = ∞; tata[u]=0
d[s] = 0

//determinăm o sortare topologică a vârfurilor
```

Drumuri minime de sursă unică în grafuri aciclice

```
s – vârful de start

//initializam distante – ca la Dijkstra
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
d[s] = 0

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

pentru fiecare u ∈ SortTop
```

Drumuri minime de sursă unică în grafuri aciclice

```
s – vârful de start

//initializam distante – ca la Dijkstra
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
d[s] = 0
//determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologica(G)

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

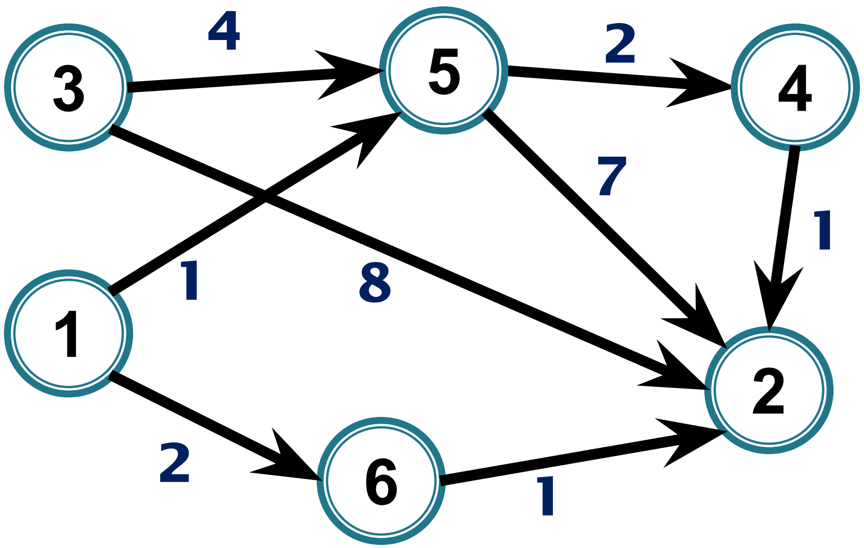
Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

//initializam distante - ca la Dijkstra
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
d[s] = 0
//determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologica(G)

pentru fiecare u ∈ SortTop
       pentru fiecare uv∈E executa
            daca d[u]+w(u,v)<d[v] atunci  //relaxam uv
                 d[v] = d[u]+w(u,v)
              tata[v] = u
```

Drumuri minime de sursă unică în grafuri aciclice

```
s – vârful de start

//initializam distante – ca la Dijkstra
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
d[s] = 0
//determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologica(G)

pentru fiecare u ∈ SortTop
       pentru fiecare uv∈E executa
            daca d[u]+w(u,v)<d[v] atunci  //relaxam uv
                  d[v] = d[u]+w(u,v)
               tata[v] = u
scrie d, tata
```

Exemplu



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

Drumuri minime de sursă unică în grafuri aciclice

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

- **Amintim algoritm**

SortTop ← ∅;

coada C ← ∅;

adauga in C toate vârfurile v cu $d^-[v]=0$

cat timp $C \neq \varnothing$ **executa**

    i ← **extrage(*C*);**

    adauga i in SortTop

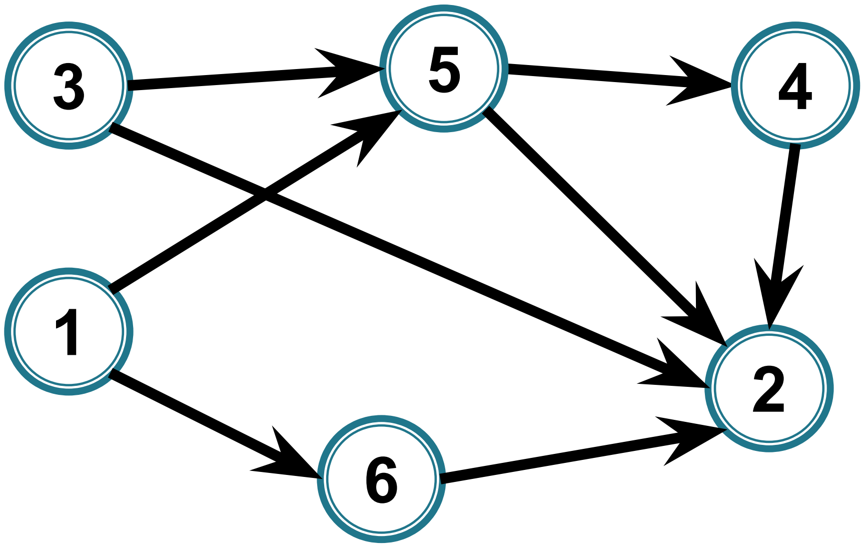
    pentru ij ∈ **E executa**

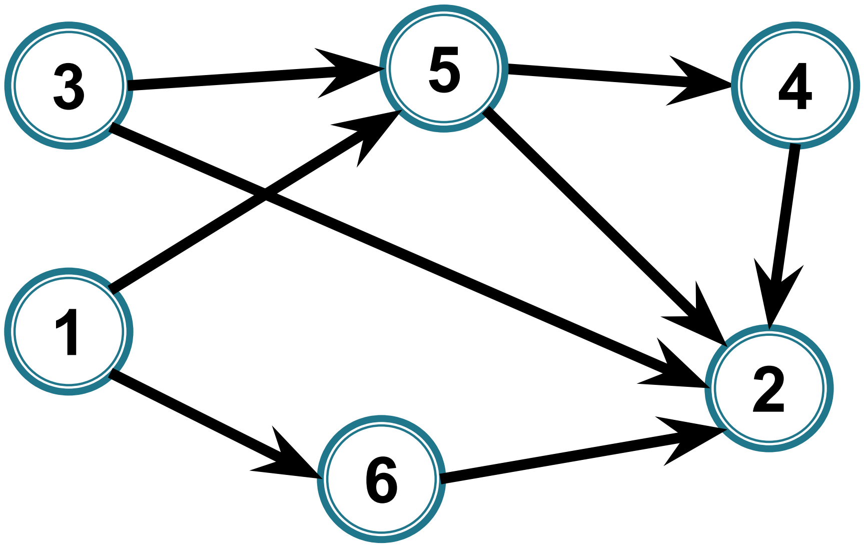
        $d^-[j] = d^-[j] - 1$

        daca $d^-[j]=0$ **atunci**

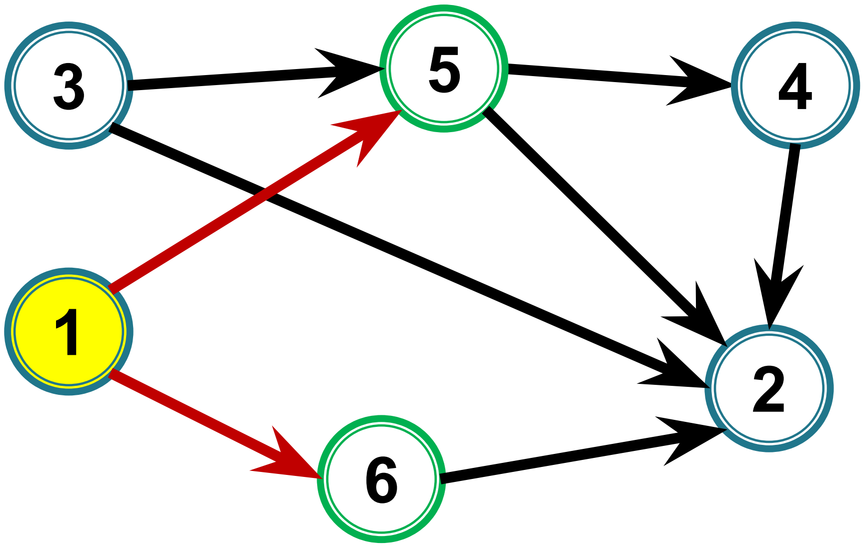
            adauga(j, C)

returneaza SortTop

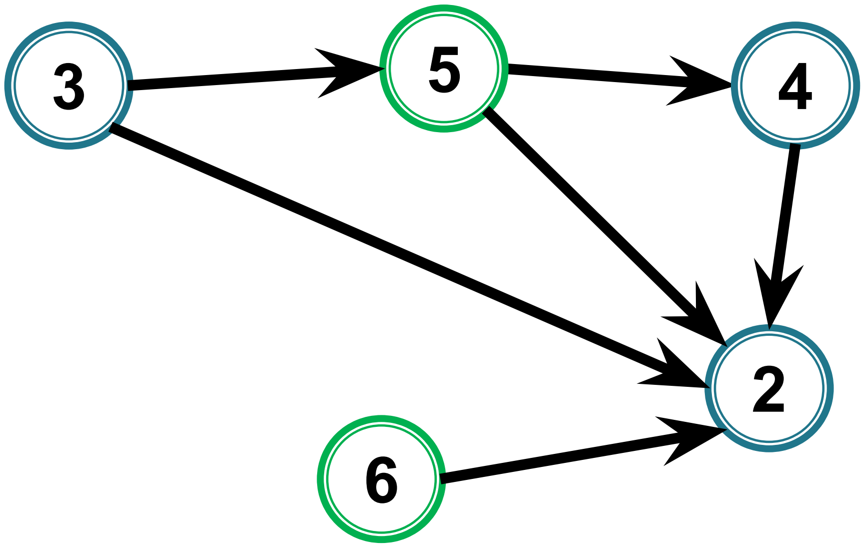

3
5
4
1
2
6



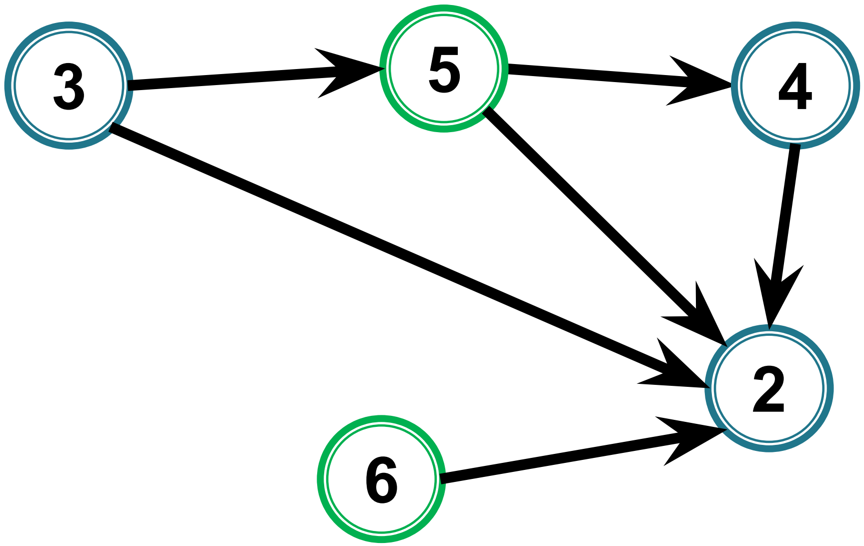
C: 1 3



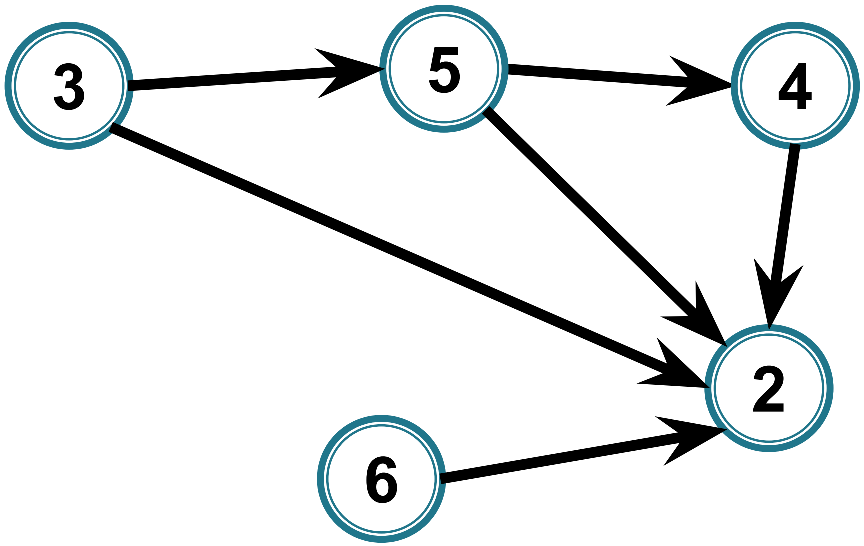
C: 1 3



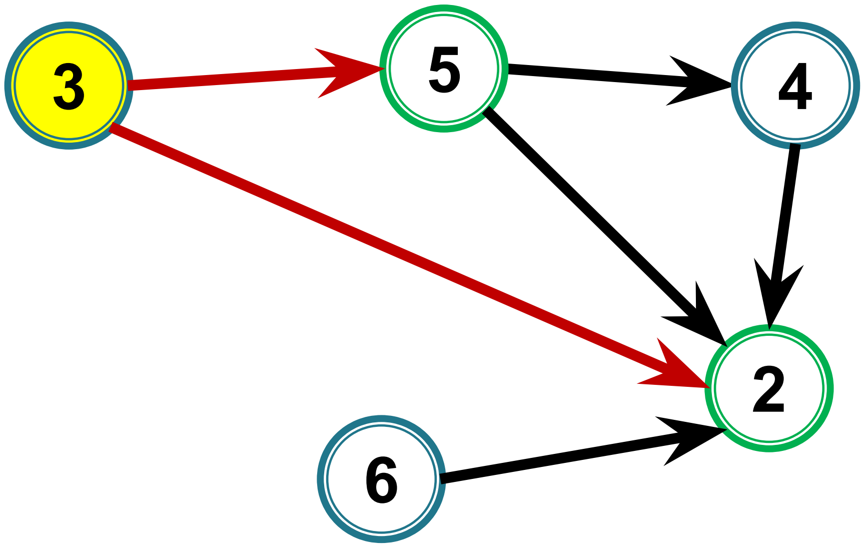
C: 1 3



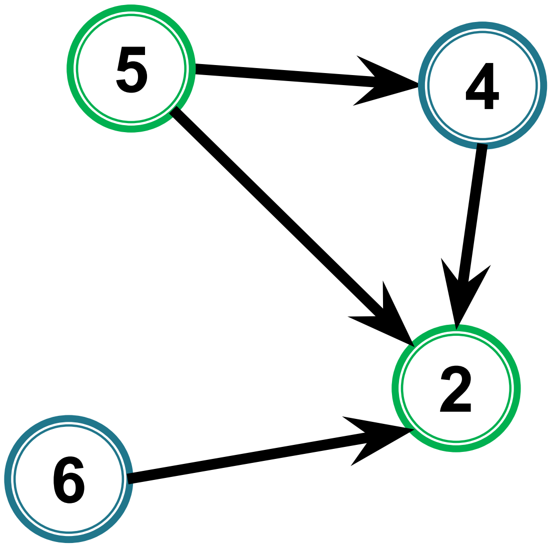
C: 1 3 6



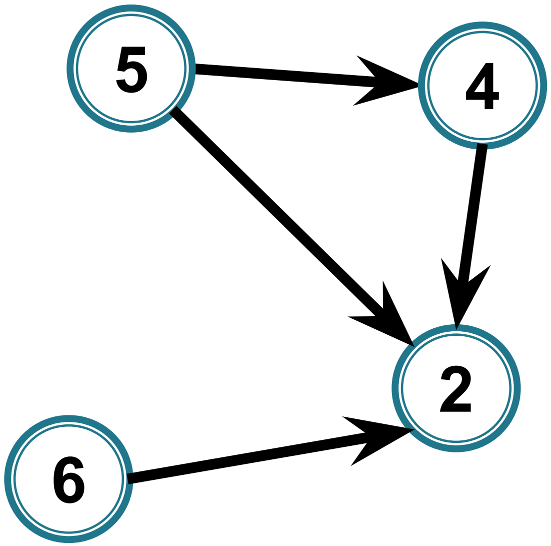
C: 1 3 6



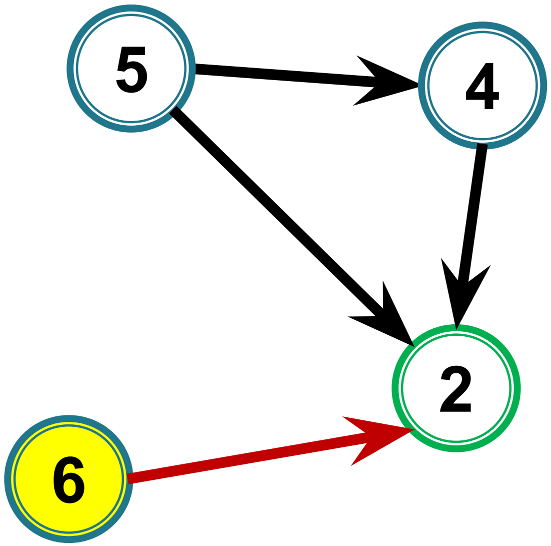
C: 1 3 6



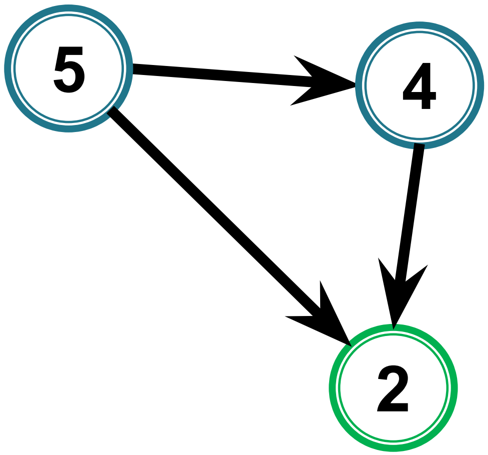
C: 1 3 6



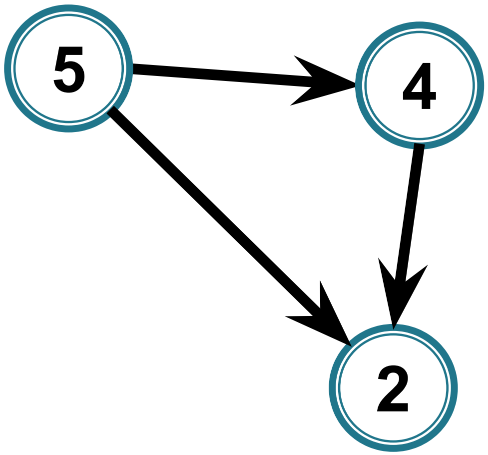
C: 1 3 6
5



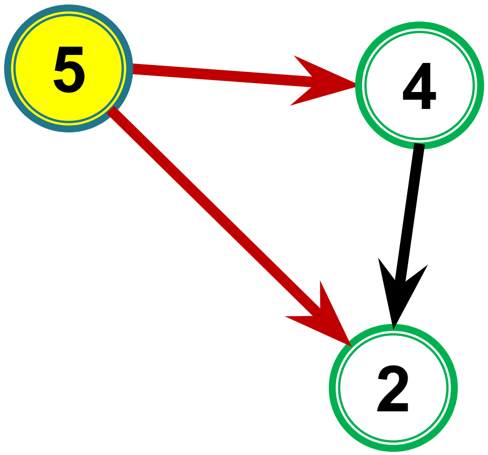
C: 1 3 6 5



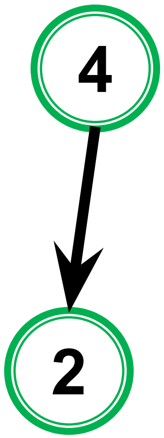
C: 1 3 6 5



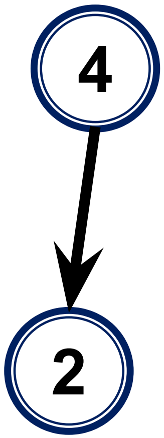
C: 1 3 6 5



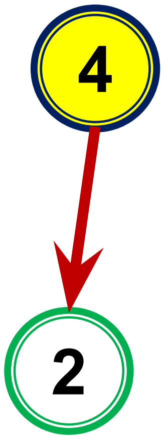
C: 1 3 6 5



C: 1 3 6 5



C: 1 3 6 5 4



C: 1 3 6 5 4

2

C: 1 3 6 5 4

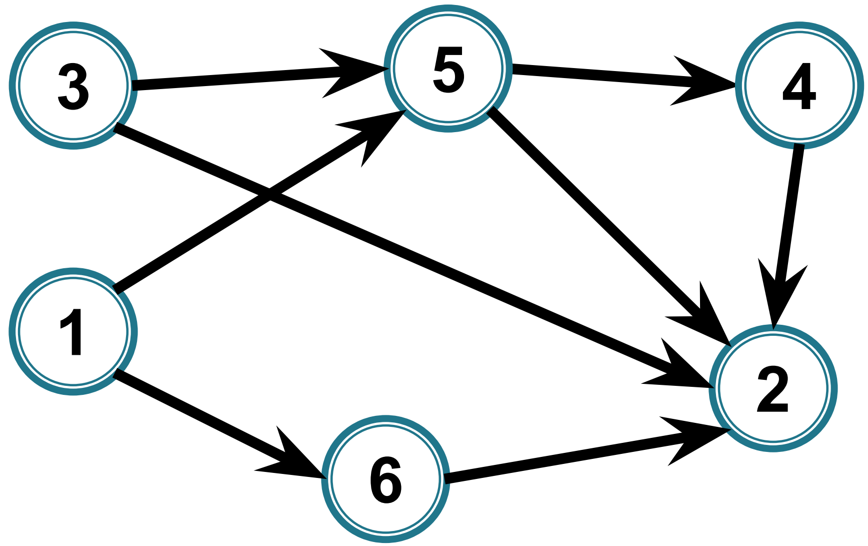
2

C: 1 3 6 5 4 2

2

C: 1 3 6 5 4 2

c: 1 3 6 5 4 2



Sortare topologică: 1 3 6 5 4 2

Sortare topologică - Algoritm

```
coada C ← ∅;
adauga in C toate vârfurile v cu d⁻[v]=0

cat timp C ≠ ∅ executa
   i ←  extrage(C);
   adauga i in sortare
   pentru ij ∈ E executa
      d⁻[j] = d⁻[j] - 1
      daca d⁻[j]=0 atunci
          adauga(j, C)
return C
```

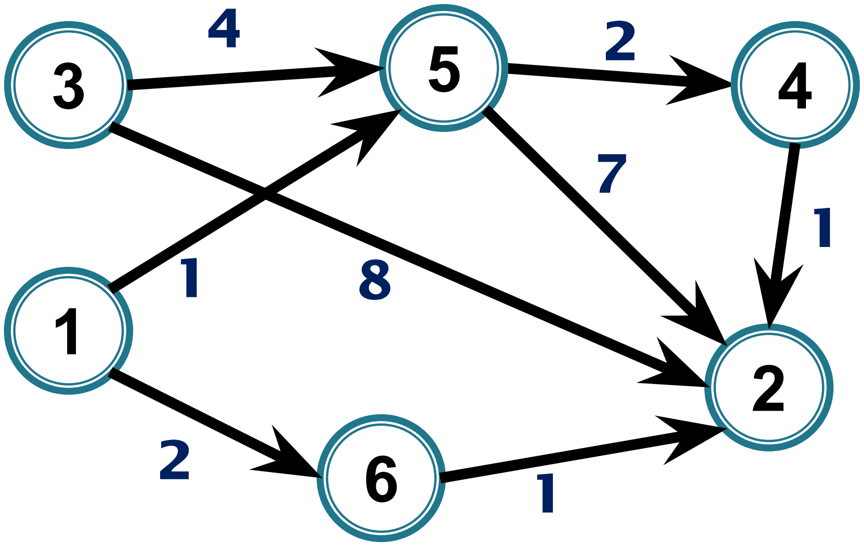
Drumuri minime de sursă unică în grafuri aciclice

- **<u>Etapa 2</u> – parcurgem vârfurile în ordinea dată de sortarea topologică și relaxăm pentru fiecare vârf arcele care ies din acesta**

Drumuri minime de sursă unică în grafuri aciclice

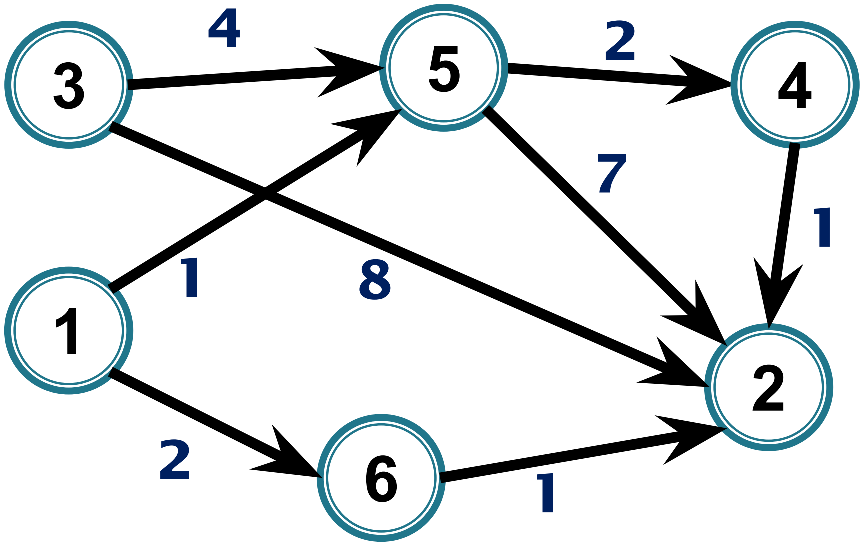
- **Inițial – determinăm o ordonare topologică a vârfurilor**
- **Amintim algoritm**

```
SortTop ← ∅;
coada C ← ∅;
adauga in C toate vârfurile v cu d⁻[v]=0

cat timp C ≠ ∅ executa
    i ←  extrage(C);
    adauga i in SortTop
    pentru ij ∈ E executa
        d⁻[j] = d⁻[j] - 1
        daca d⁻[j]=0 atunci
            adauga(j, C)

returneaza SortTop
```



Sortare topologică

1, 3, 6, 5, 4, 2



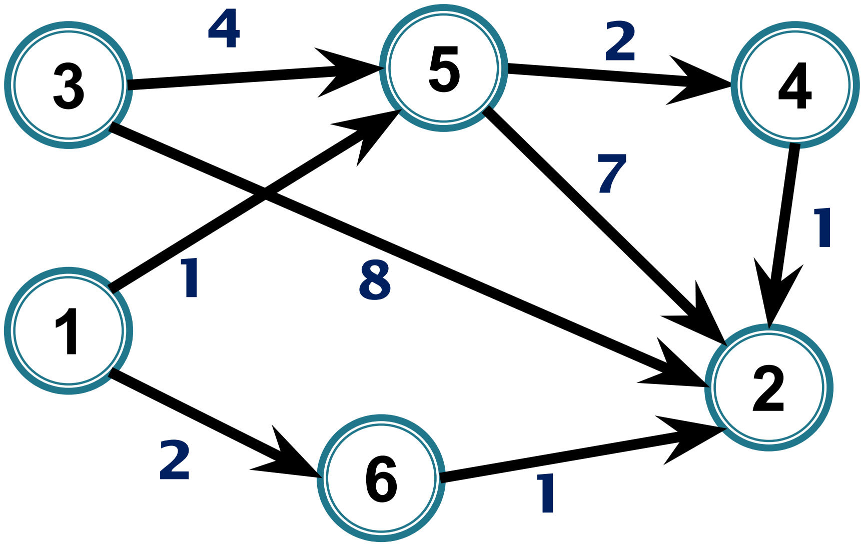
Sortare topologică

1, 3, 6, 5, 4, 2

s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

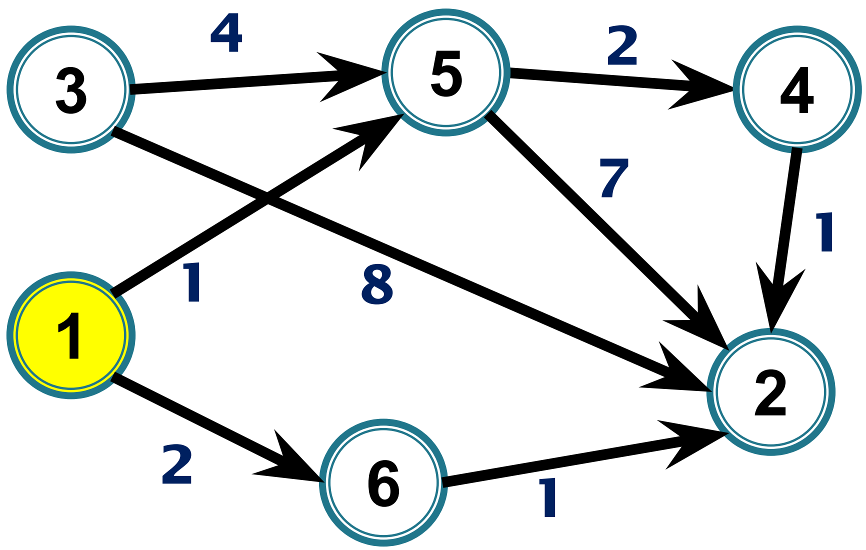


Sortare topologică 1, 3, 6, 5, 4, 2

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]



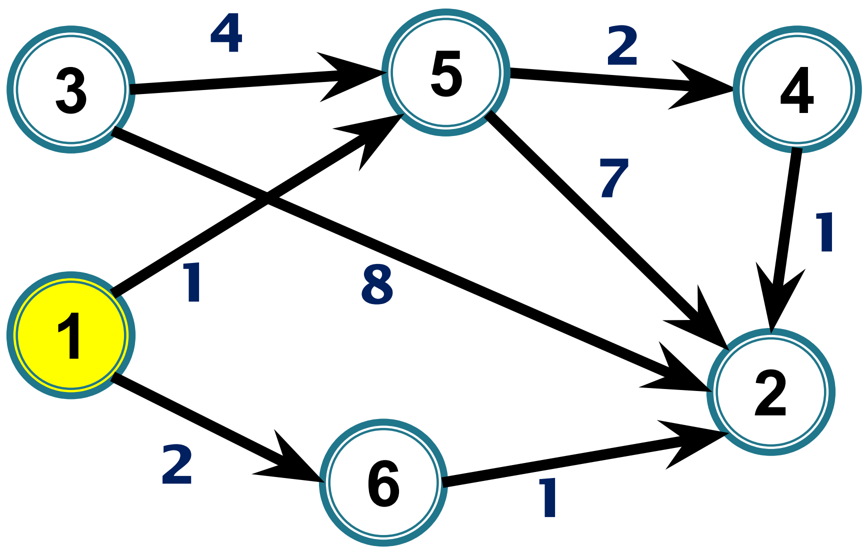
Sortare topologică
1, 3, 6, 5, 4, 2

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

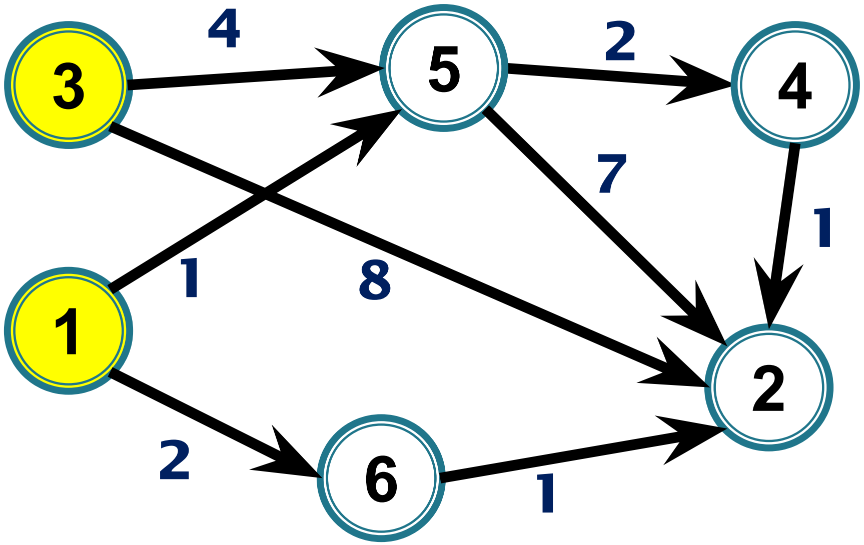
s=3 – vârf de start

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, puteam să nu îl considerăm (să ignorăm vârfurile din ordonare topologică aflate înaintea lui s)

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



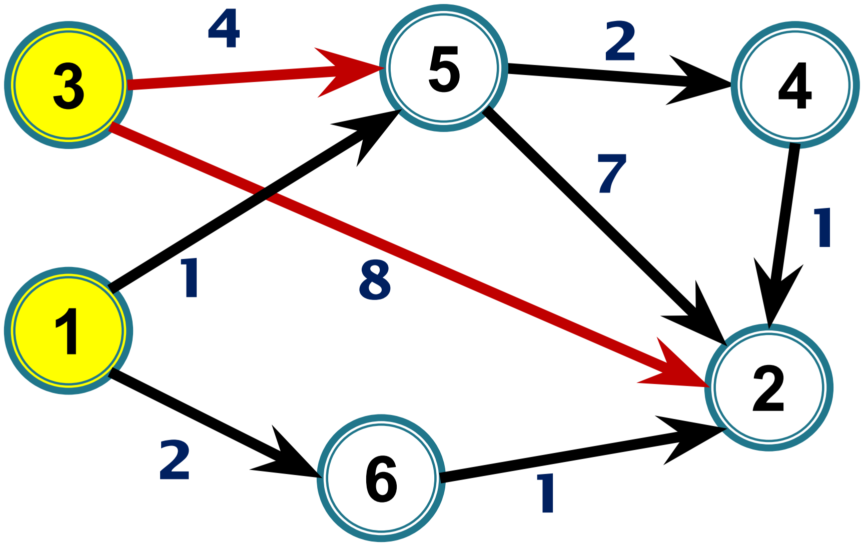
Sortare topologică 1, 3, 6, 5, 4, 2

s=3 – vârf de start

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

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

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



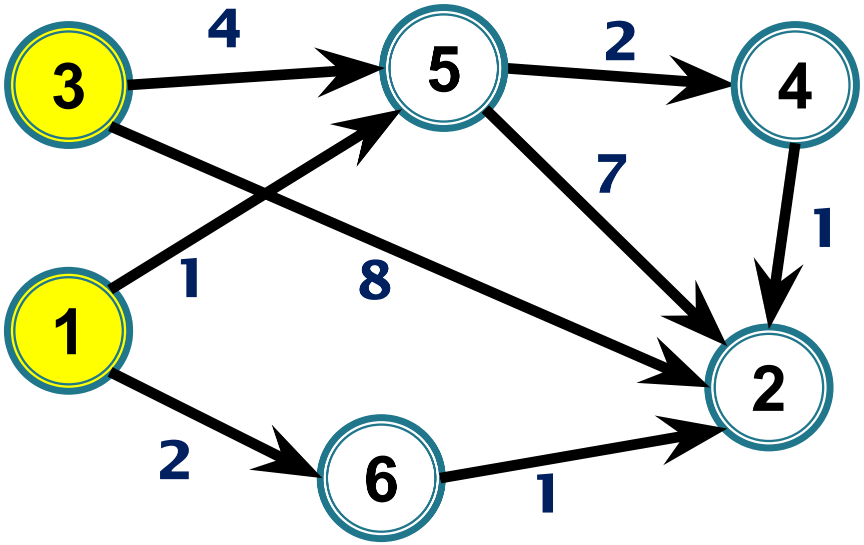
Sortare topologică 1, 3, 6, 5, 4, 2

s=3 – **vârf de start**

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

d/tata	1	2	3	4	5	6
	[∞/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)}`



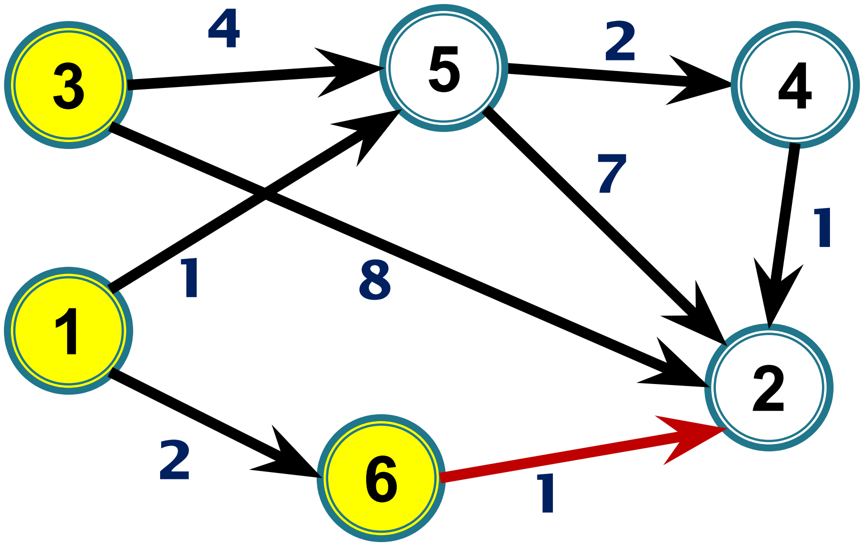
Sortare topologică: 1, 3, 6, 5, 4, 2

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)}
```



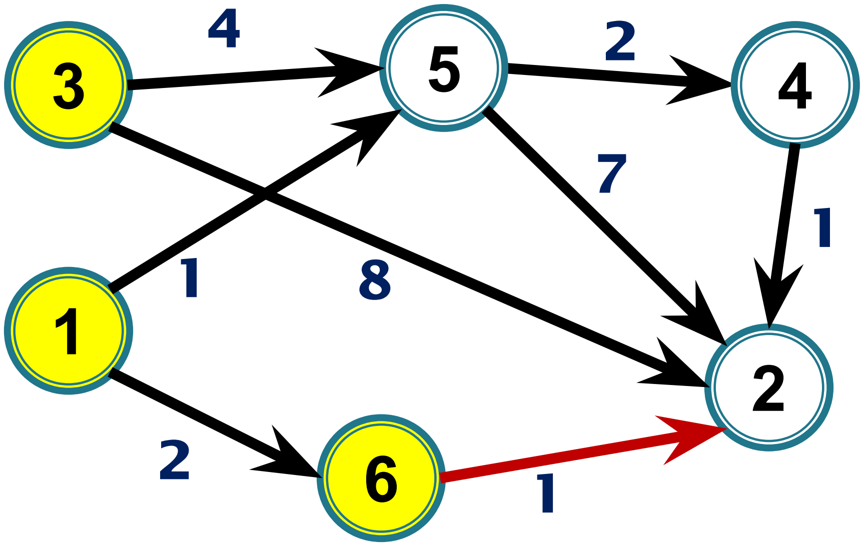
Sortare topologică 1, 3, 6, 5, 4, 2

s=3 – vârf de start

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

d/tata	1	2	3	4	5	6
	[∞/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)}`



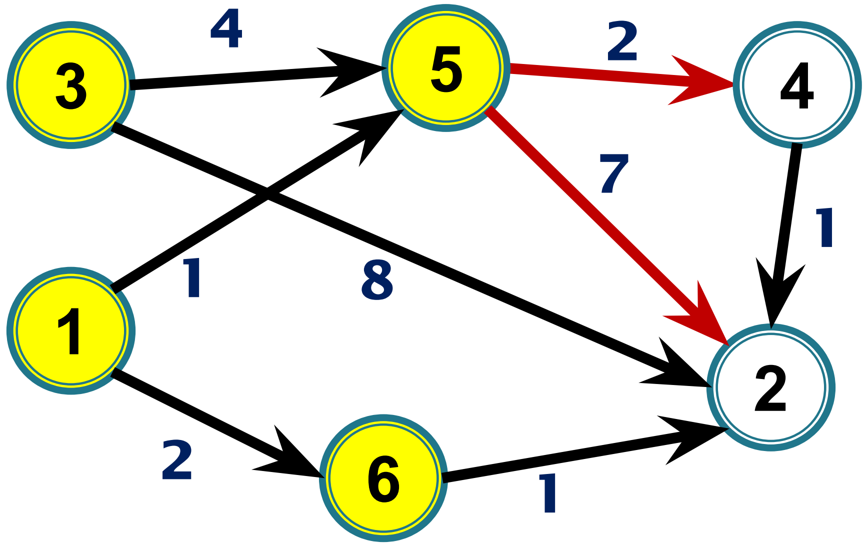
Sortare topologică: 1, 3, 6, 5, 4, 2

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)}`



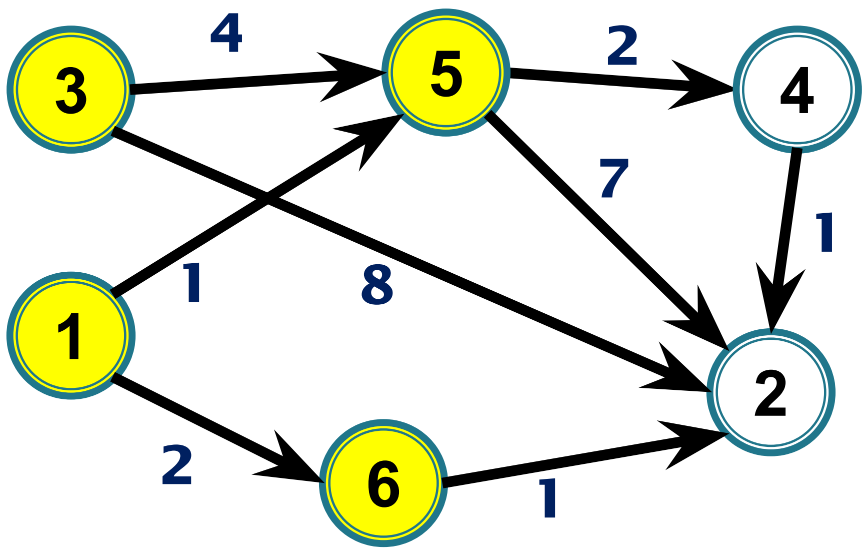
Sortare topologică 1, 3, 6, 5, 4, 2

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)}



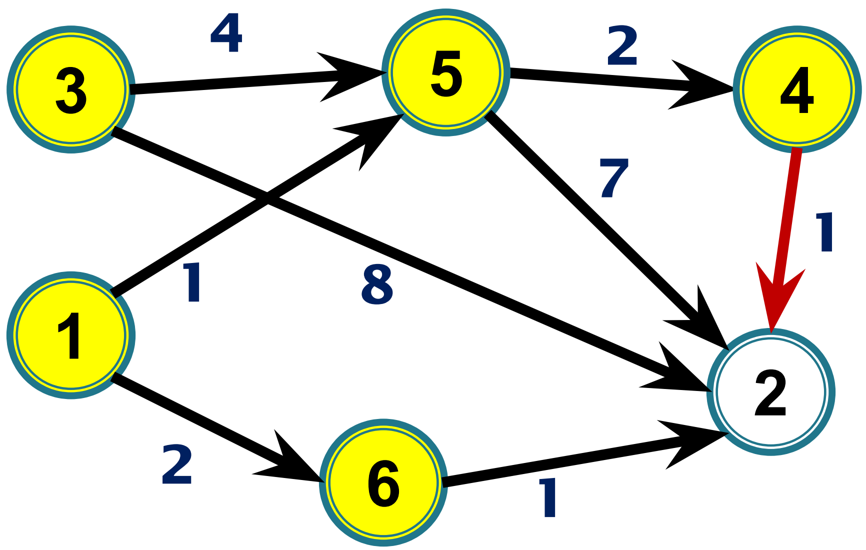
Sortare topologică: 1, 3, 6, 5, 4, 2

s=3 – vârf de start

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

d/tata	1	2	3	4	5	6
	[∞/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)}
```



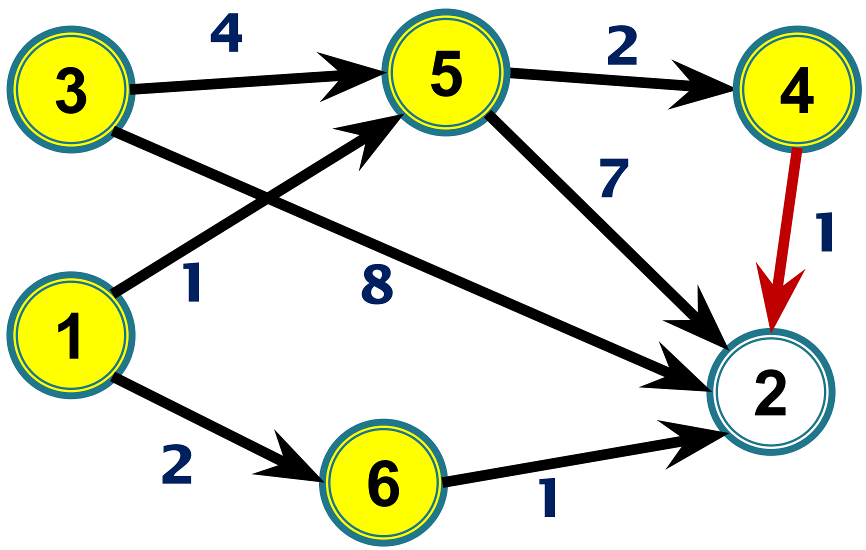
Sortare topologică 1, 3, 6, 5, 4, 2

s=3 – **vârf de start**

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

d/tata	1	2	3	4	5	6
	[∞/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)}`



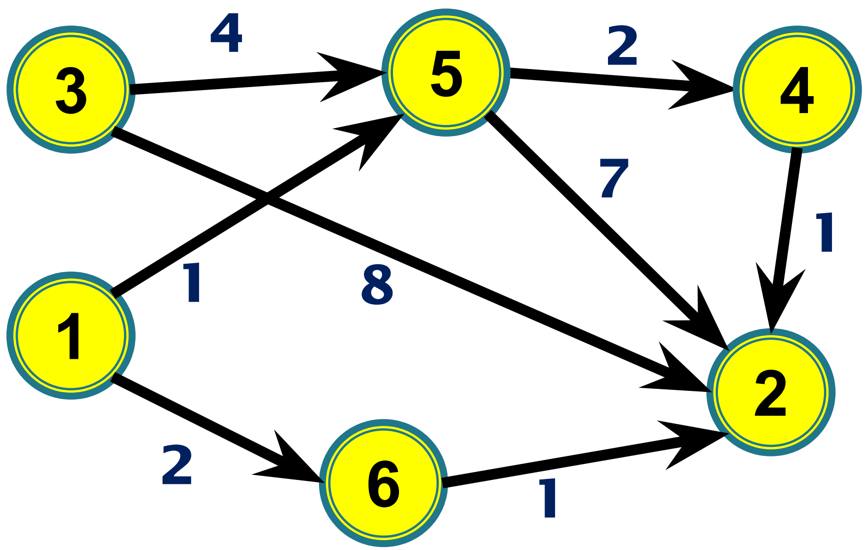
Sortare topologică: **1, 3, 6, 5, 4, 2**

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)}`



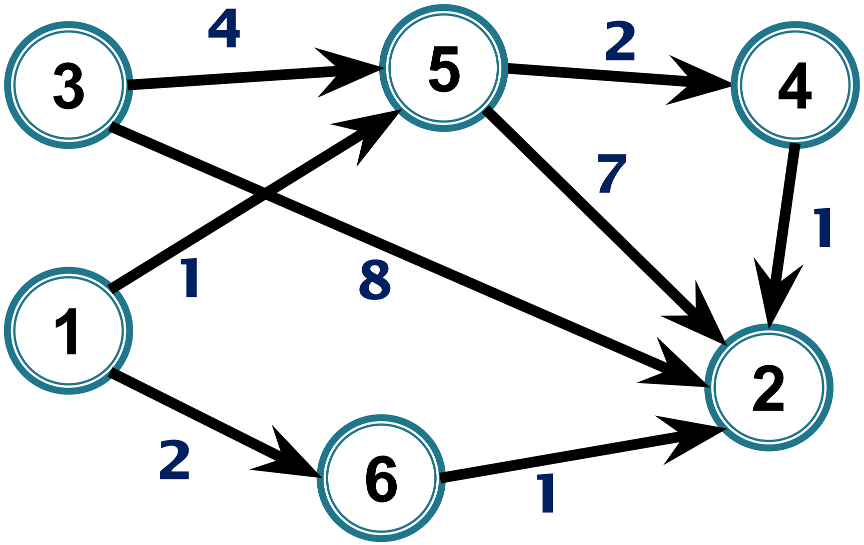
Sortare topologică 1, 3, 6, 5, 4, 2

s=3 – vârf de start

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

d/tata	1	2	3	4	5	6
	[∞/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)}`

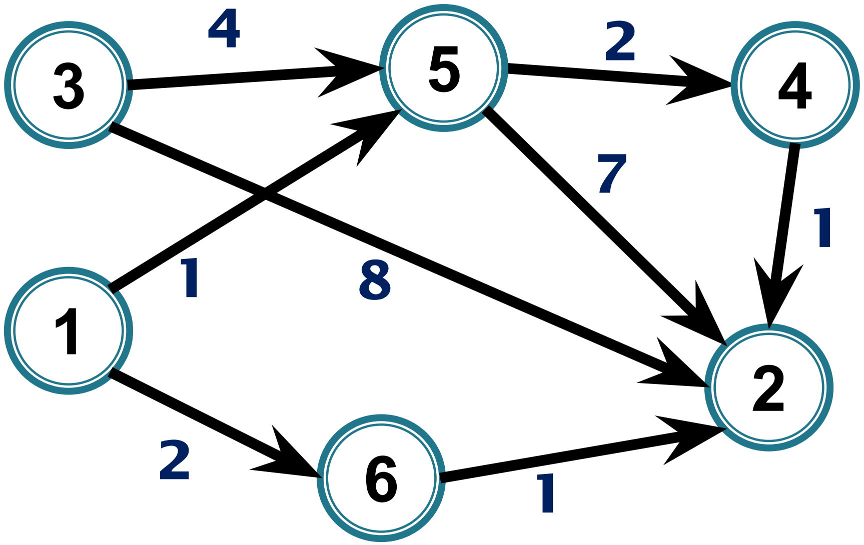


Sortare topologică: 1, 3, 6, 5, 4, 2

s=3 – vârf de start

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

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

Soluție

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();
   adauga u in sortare
 }
```

Drumuri minime de sursă unică în grafuri aciclice

```
s - vârful de start

//initializam distante - ca la Dijkstra
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
d[s] = 0
//determinăm o sortare topologică a vârfurilor
SortTop = sortare_topologica(G)

pentru fiecare u ∈ SortTop
       pentru fiecare uv∈E executa
            daca d[u]+w(u,v)<d[v] atunci  //relaxam uv
                 d[v] = d[u]+w(u,v)
               tata[v] = u
scrie d, tata
```

Drumuri minime de sursă unică în grafuri aciclice

Complexitate

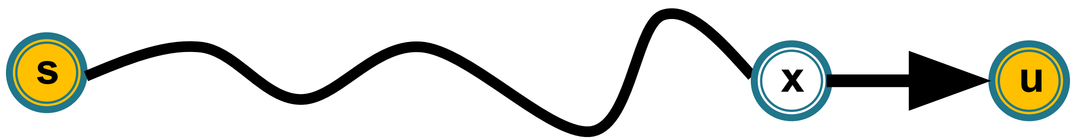
- **Inițializare** -> O(n)
- **Sortare topologică** -> O(m+n)
- **m * relaxare uv** -> O(m)

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

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