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

Drumuri minime de sursă unică în grafuri aciclice

- **Ipoteze:**
  - Graful <u>nu</u> conţine circuite
  - Arcele pot avea <u>şi cost negativ</u>

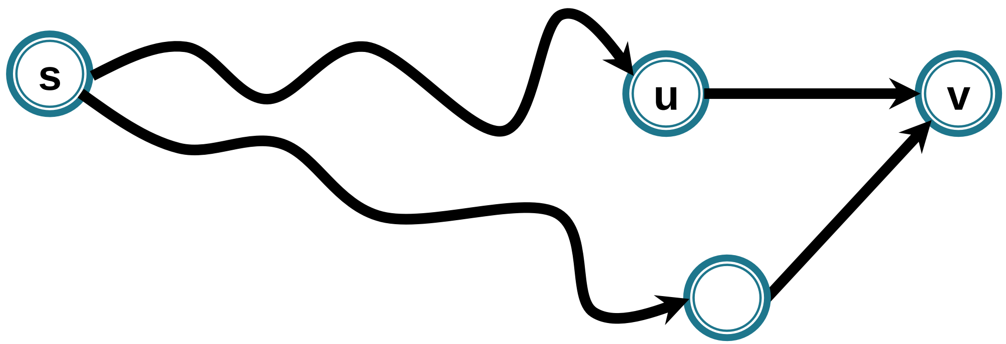
Drumuri minime de sursă unică în grafuri aciclice

- **Amintim:**

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

  - atunci putem calcula distanţele după relaţia

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

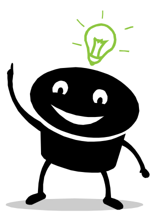


Drumuri minime de sursă unică în grafuri aciclice

- **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$\in$E $\Rightarrow$

  - Ar fi utilă o ordonare a vârfurilor astfel încât dacă uv$\in$E, atunci u se află înaintea lui v



O astfel de ordonare există dacă graful nu conţine circuite = sortarea topologică

Sortare topologică

Sortare topologică

- **Fie G = (V, E) graf orientat**
- **Sortare topologică a lui G =**

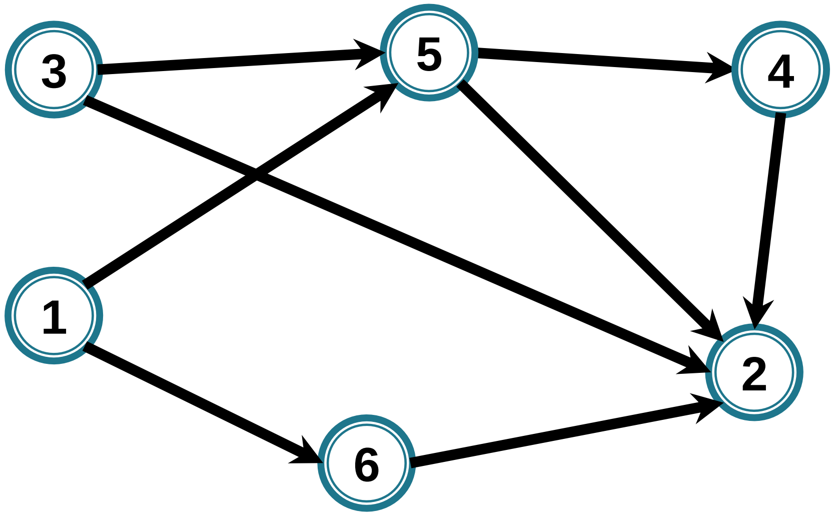
  ordonare a vârfurilor astfel încât dacă uv $\in$ E atunci u se află înaintea lui v în ordonare

  - **Nu este neapărat unică**

Sortare topologică

- **Fie G = (V, E) graf orientat**
- **Sortare topologică a lui G =**

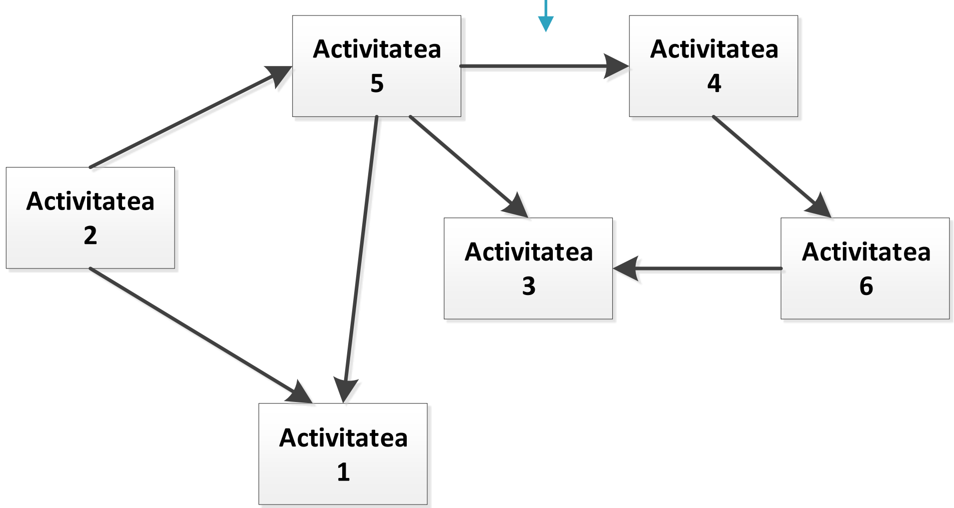
  ordonare a vârfurilor astfel încât dacă uv $\in$ E atunci u se află înaintea lui v în ordonare



Sortare topologică

- **Aplicații**
  - Ordinea de calcul în proiecte în care intervin relații de dependență / precedență (exp: calcul de formule, ordinea de compilare când clasele/pachetele depind unele de altele)
  - Detecție de deadlock
  - Determinarea de drumuri critice

Activitatea 4 depinde de 5, deci trebuie să se desfășoare după ea



	A	B	C	D
1		3	2	
2		3	6	0
3				
4		"=B1+D2"	"=2*B2"	"=2*C1+C2"

formulele din celulele B2..D2

În ce ordine trebuie executate activitățile?

În ce ordine se evaluează formulele?

Probleme – dacă există dependențe circulare

Sortare topologică

- **Fie G = (V, E) graf orientat**
- **Sortare topologică a lui G =**

  ordonare a vârfurilor astfel încât dacă uv $\in$ E atunci u se află înaintea lui v în ordonare

- **Propoziţie.** Dacă G este aciclic atunci G are o sortare topologică

Sortare topologică

- **Fie G = (V, E) graf orientat**
- **Sortare topologică a lui G =**

  ordonare a vârfurilor astfel încât dacă uv $\in$ E atunci u se află înaintea lui v în ordonare

- **Propoziţie.** Dacă G este aciclic atunci G are o sortare topologică
  - **Demonstrație $\Rightarrow$ Algoritm?**

Care vârf poate fi primul în sortarea topologică?

Sortare topologică - Algoritm

- **Fie G = (V, E) graf orientat**

- **Lemă.** Dacă G este aciclic, atunci G are cel puţin un vârf v cu gradul intern 0 ($d^-(v) = 0$).

Sortare topologică – Algoritm

- **Fie G = (V, E) graf orientat**
- **Lemă.** Dacă G este aciclic, atunci G are cel puţin un vârf v cu gradul intern 0 ($d^-(v) = 0$).

Demonstrație: considerăm un drum elementar maxim. Extremitatea inițială a sa are grad intern 0

(v. si dem. Proprietății: un arbore cu $n > 2$ vârfuri are minim 2 vârfuri terminale)

Sortare topologică – Algoritm

- **Fie G = (V, E) graf orientat**

- **Lemă.** Dacă G este aciclic, atunci G are cel puţin un vârf v cu gradul intern 0 ($d^-(v) = 0$).

- **Algoritm**

```
cât timp |V(G)|>0 execută
      alege v cu d⁻(v) = 0
      adauga v in ordonare
      G ← G - v
```

- **Corectitudinea** – rezultă din **Lemă** + inducție

Pseudocod

Sortare topologică – Algoritm

- Algoritm

cât timp |V(G)|>0 execută

alege v cu $d^-(v) = 0$

adauga v in ordonare

G $\leftarrow$ G - v

Implementare? Complexitate?

Sortare topologică – Algoritm

- Algoritm

```
cât timp |V(G)|>0 execută
      alege v cu d⁻(v) = 0
      adauga v in ordonare
      G ← G - v
```

- Implementare – similar BF
    - **Pornim cu <u>toate</u> vârfurile cu grad intern 0 și le adăugăm într-o coadă**
    -

Sortare topologică – Algoritm

- Algoritm

```
cât timp |V(G)|>0 execută
      alege v cu d⁻(v) = 0
      adauga v in ordonare
      G ← G - v
```

- Implementare – similar BF
  - **Pornim cu toate vârfurile cu grad intern 0 și le adăugăm într-o coadă**
  - **Repetăm:**

    - extragem un vârf din coadă

    - îl eliminăm din graf (= scădem gradele interne ale vecinilor, nu îl eliminăm din reprezentare)

    -

Sortare topologică – Algoritm

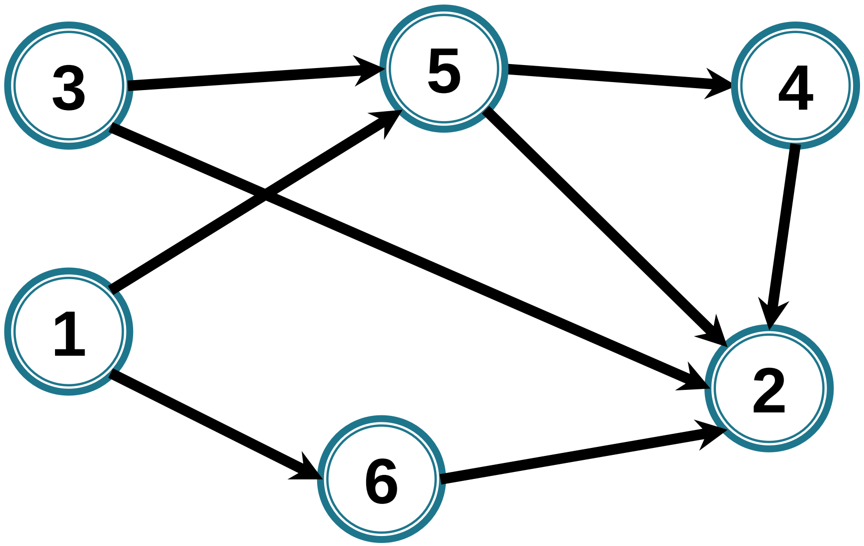
- **Algoritm**

```
cât timp |V(G)|>0 execută
        alege v cu d⁻(v) = 0
        adauga v in ordonare
        G ← G - v
```

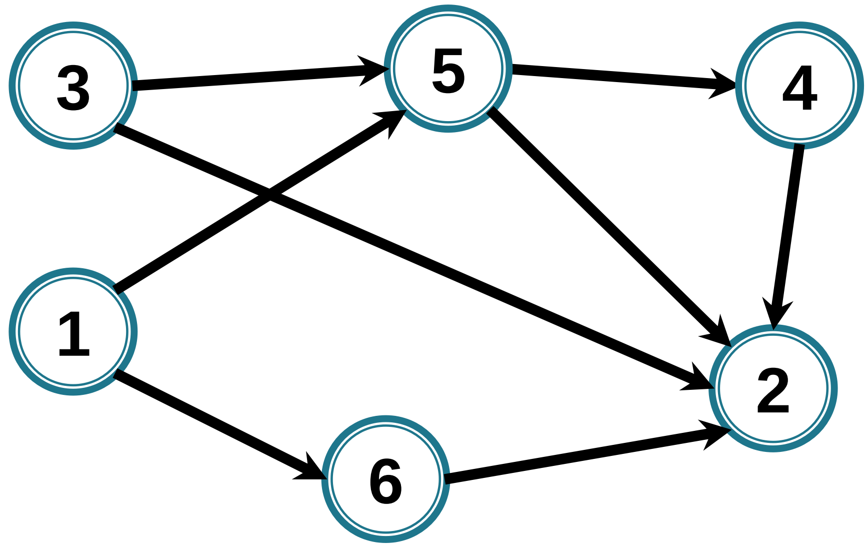
- **Implementare – similar BF**
  - Pornim cu toate vârfurile cu grad intern 0 și le adăugăm într-o coadă
  - Repetăm:
    - extragem un vârf din coadă
    - îl eliminăm din graf (= scădem gradele interne ale vecinilor, nu îl eliminăm din reprezentare)
    - adăugăm în coadă vecinii al căror grad intern devine 0

Exemplu

Sortare topologică

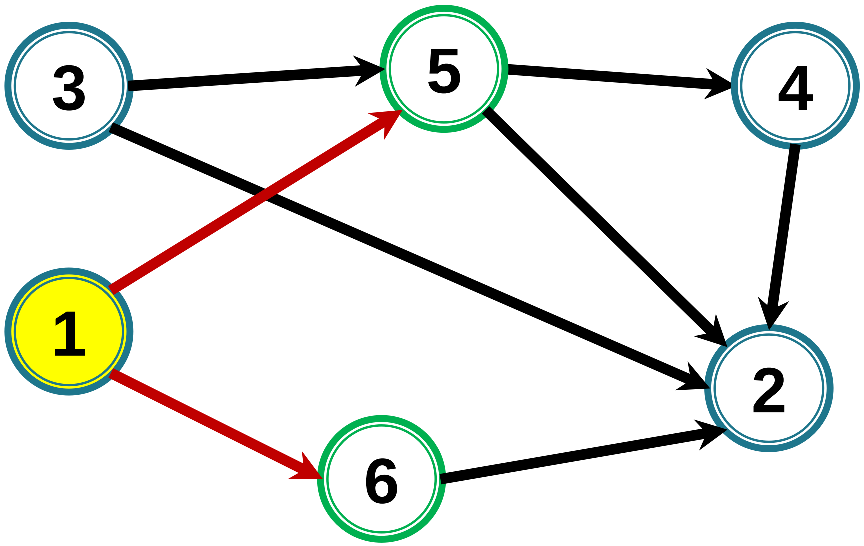


Sortare topologică



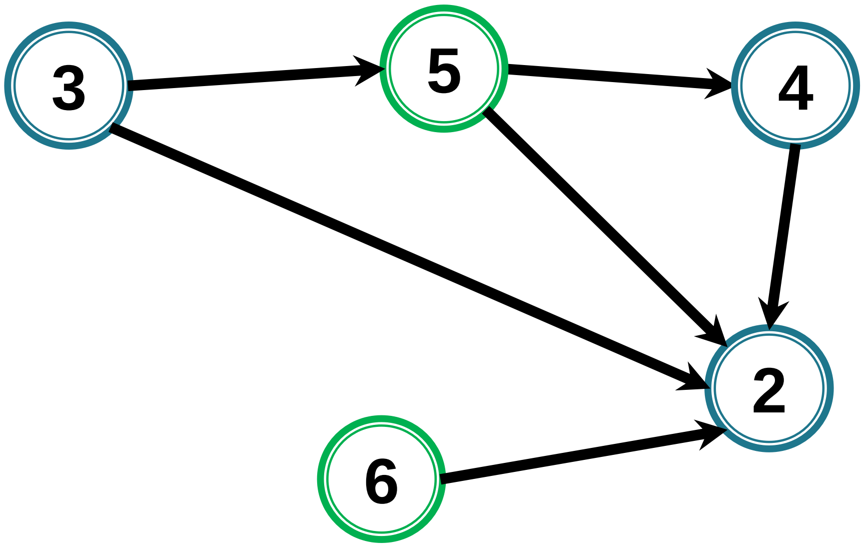
C: 1 3

Sortare topologică



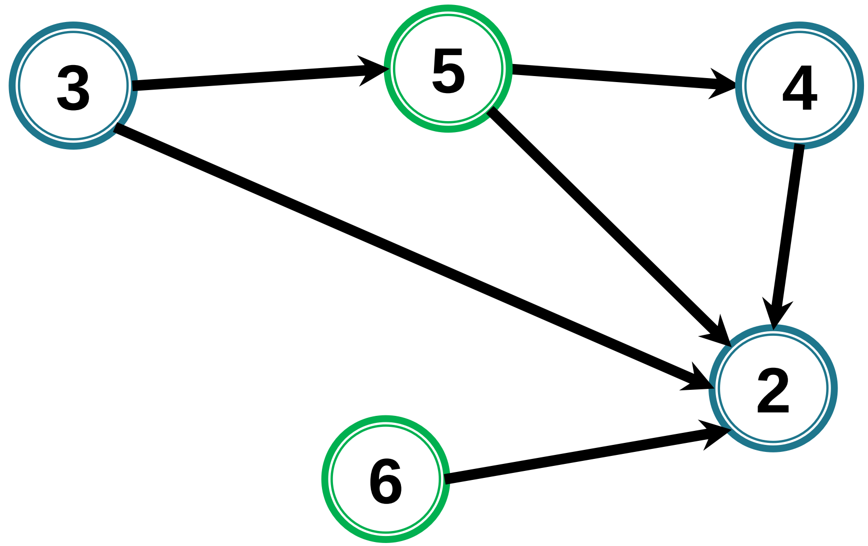
C: 1 3

Sortare topologică



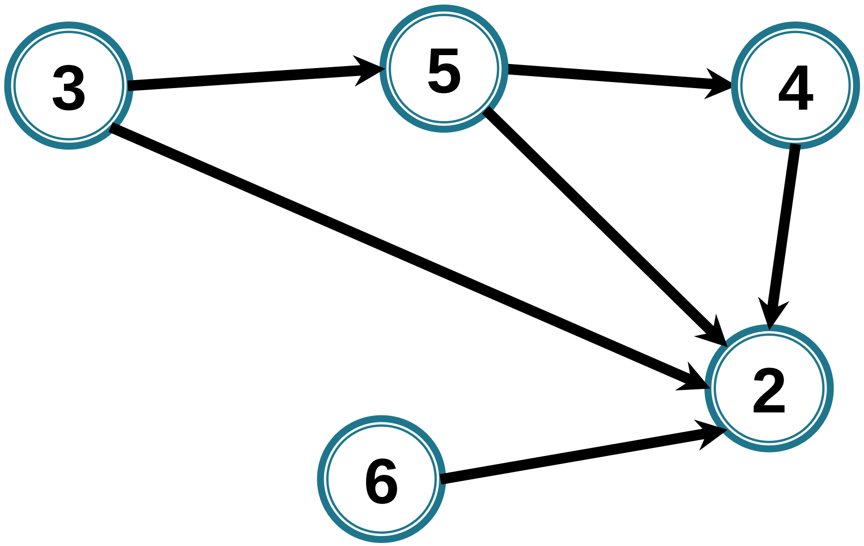
C: 1 3

Sortare topologică



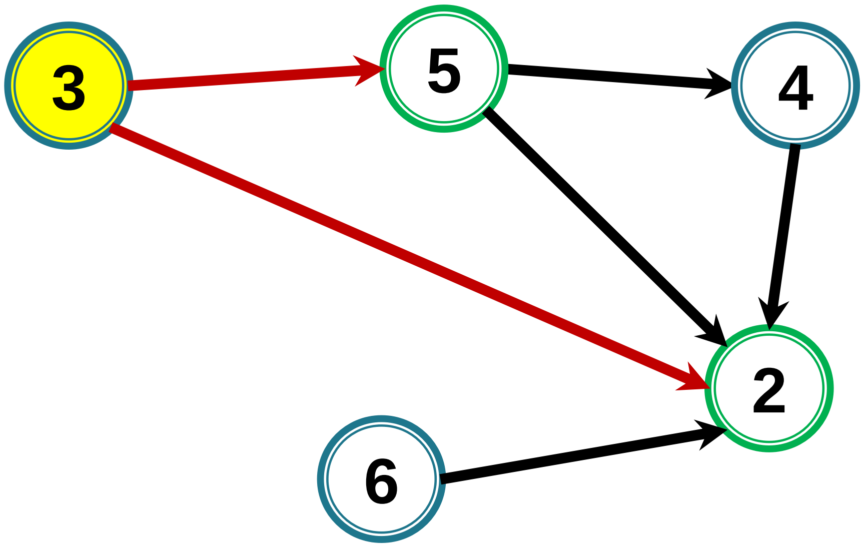
C: 1 3 6

Sortare topologică



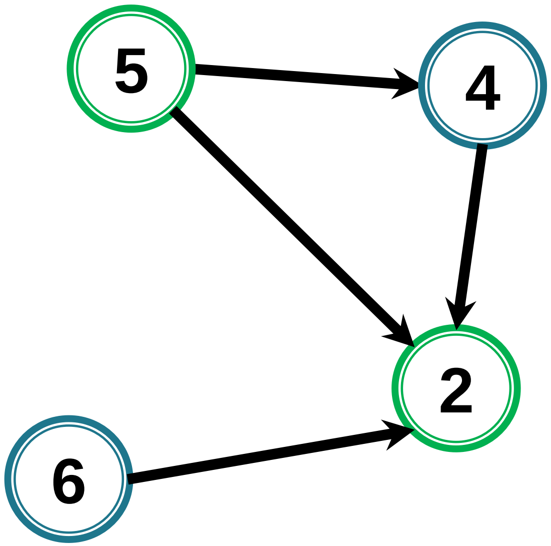
C: 1 3 6

Sortare topologică



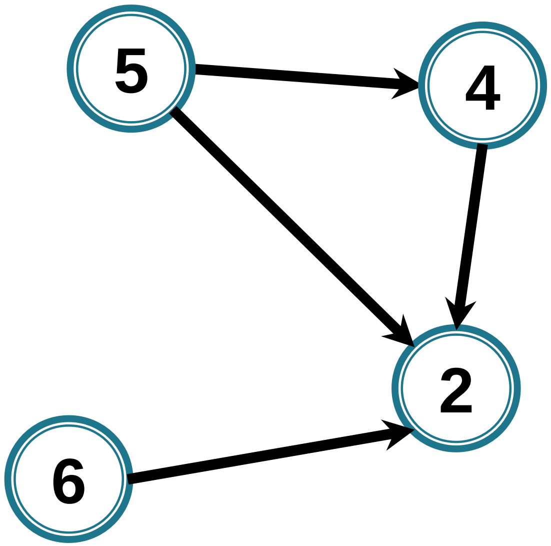
C: 1 3 6

Sortare topologică



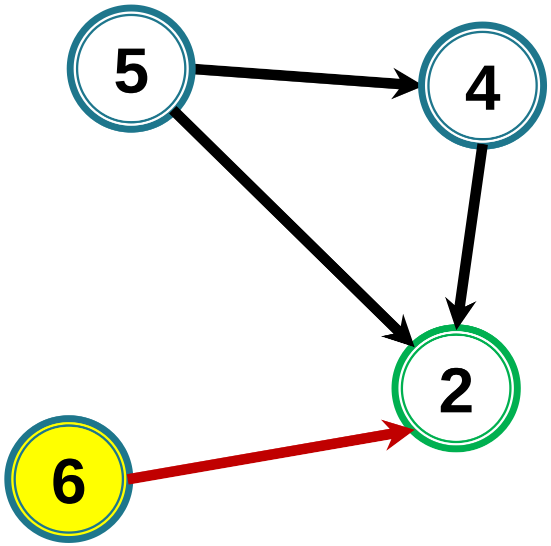
C: 1 3 6

Sortare topologică



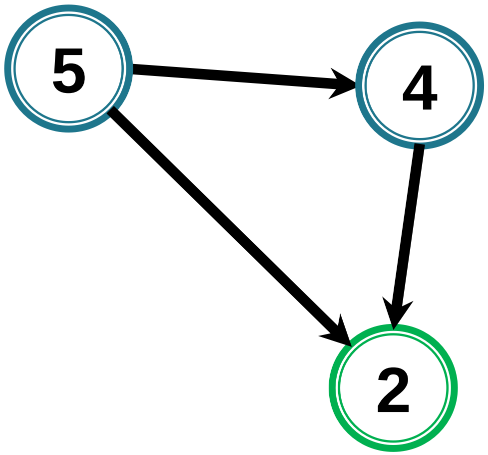
C: 1 3 6 5

Sortare topologică



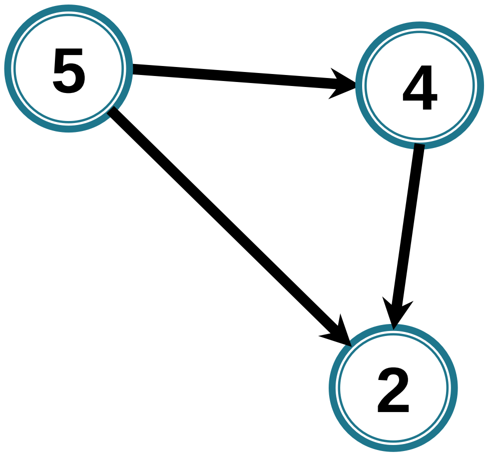
C: 1 3 6 5

Sortare topologică



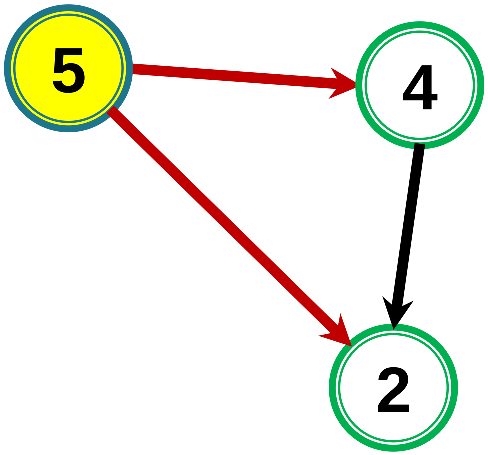
C: 1 3 6 5

Sortare topologică



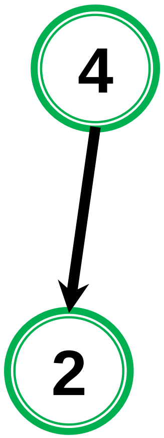
C: 1 3 6 5

Sortare topologică



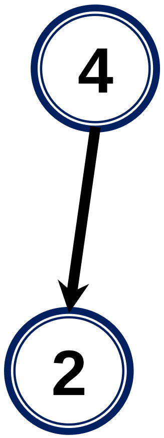
C: 1 3 6 5

Sortare topologică



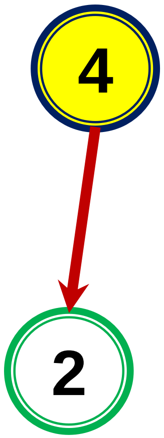
C: 1 3 6 5

Sortare topologică



C: 1 3 6 5 4

Sortare topologică



C: 1 3 6 5 4

Sortare topologică

C: 1 3 6 5 4

Sortare topologică

C: 1 3 6 5 4 2

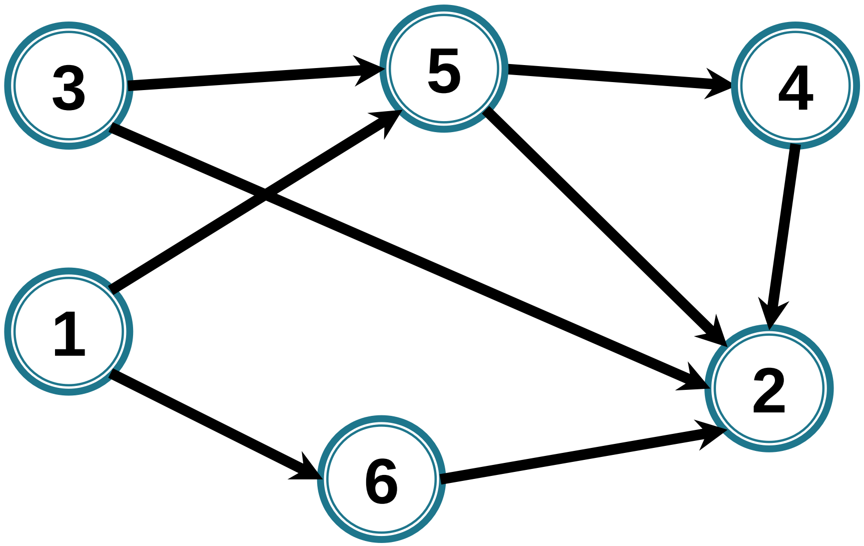
Sortare topologică

C: 1 3 6 5 4 2

Sortare topologică

C: 1 3 6 5 4 2

Sortare topologică



Sortare topologică: 1 3 6 5 4 2

Sortare topologică – Algoritm

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

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
```

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
```

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)
```

Sortare topologică

- Ce se întâmplă dacă graful conține totuși circuite?
- Cum detectăm acest lucru pe parcursul algoritmului?

Alt algoritm

Sortare topologică - Alt algoritm

- Există un algoritm bazat pe DF, pornind de la următoarea **observație:**
  - Dacă **fin**[u] = momentul la care a fost **finalizat** vârful u în parcurgerea DF avem:

    $$uv \in E \Rightarrow fin[u] > fin[v]$$

  -

Sortare topologică – Alt algoritm

- Există un algoritm bazat pe DF, pornind de la următoarea **observație:**
  - Dacă **fin**[u] = momentul la care a fost **finalizat** vârful u în parcurgerea DF avem:

    $$uv \in E \Rightarrow fin[u] > fin[v]$$

  - Atunci sortare descrescătoare în raport cu final => sortare topologică

Sortare topologică - Alt algoritm

- Există un algoritm bazat pe DF, pornind de la următoarea **observație:**

  - Dacă **fin**[u] = momentul la care a fost **finalizat** vârful u în parcurgerea DF avem:

    $$uv \in E \Rightarrow fin[u] > fin[v]$$

  - Atunci sortare descrescătoare în raport cu final => sortare topologică

    $\Rightarrow$ **Idee algoritm:** **Memorăm vârfurile într-o stivă pe măsura finalizării lor**; ordinea în care sunt scoase din stivă = sortare topologică

Sortare topologică – Alt algoritm

```
Stack S;

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);
```

Sortare topologică – Alt algoritm

```
Stack S;

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 DAG
(fără circuite)

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
```

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
```

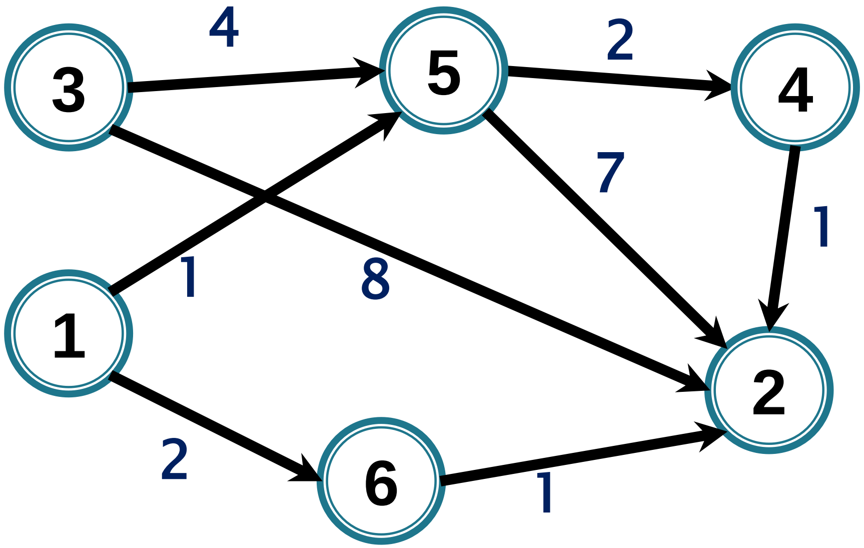
Drumuri minime de sursă unică în grafuri aciclice

```
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
```

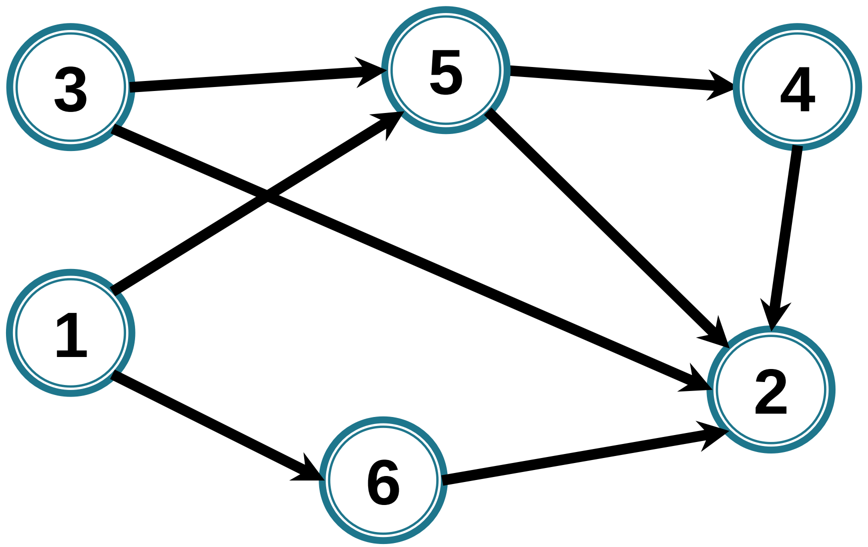
Exemplu



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

Drumuri minime de sursă unică în grafuri aciclice

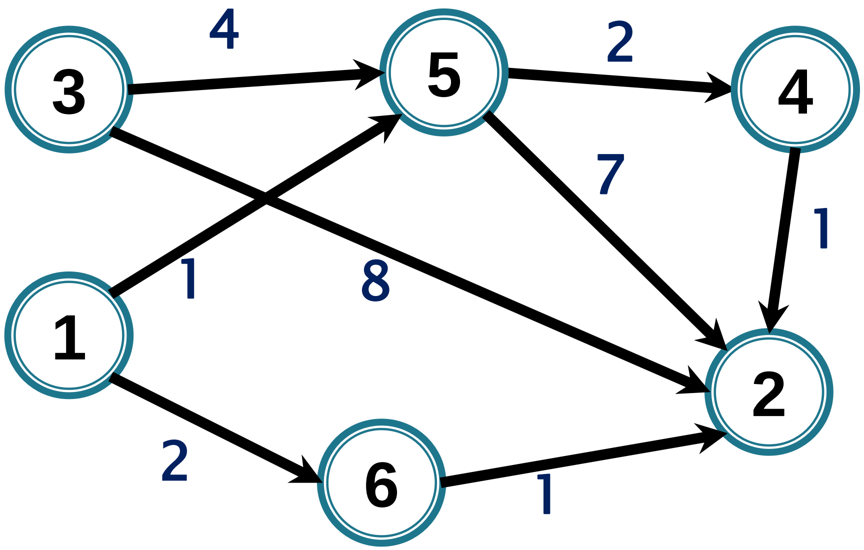
- <u>Etapa 1</u> – determinăm o ordonare topologică a vârfurilor



Sortare topologică: 1 3 6 5 4 2

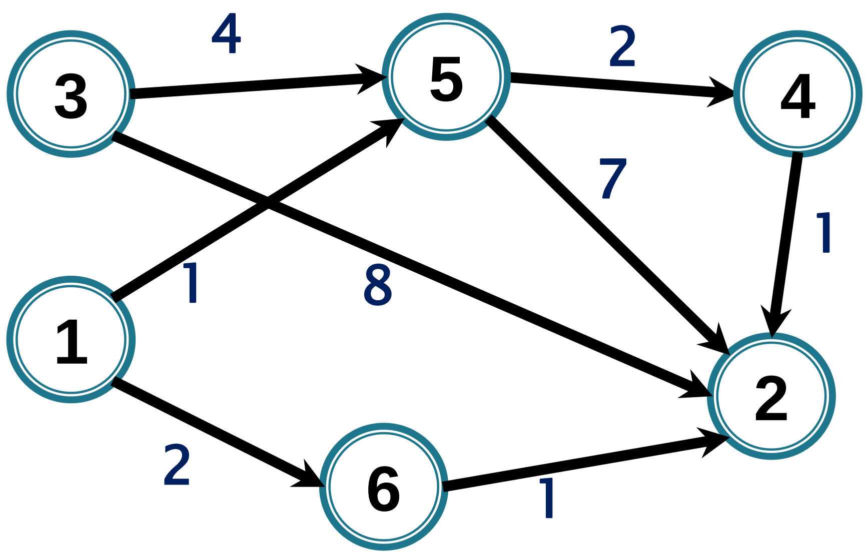
Drumuri minime de sursă unică în grafuri aciclice

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



Sortare topologică

1, 3, 6, 5, 4, 2



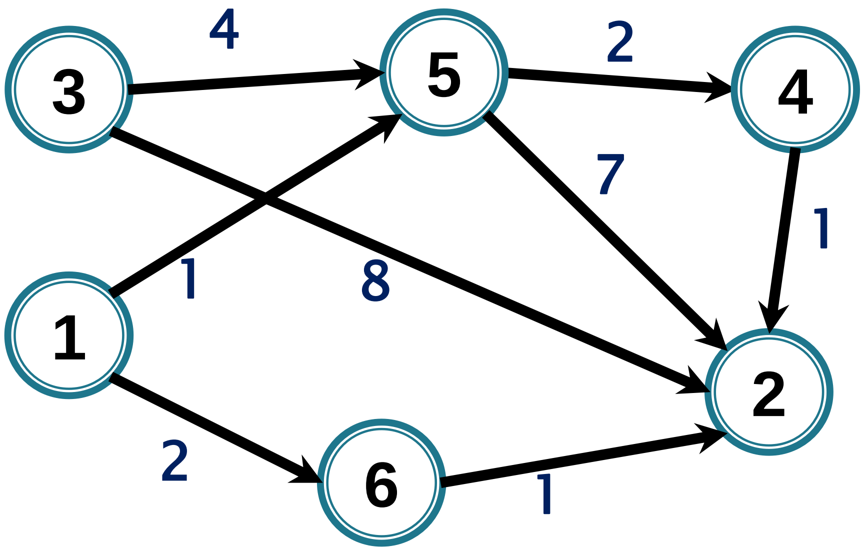
Sortare topologică

1, 3, 6, 5, 4, 2

s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2



Sortare topologică

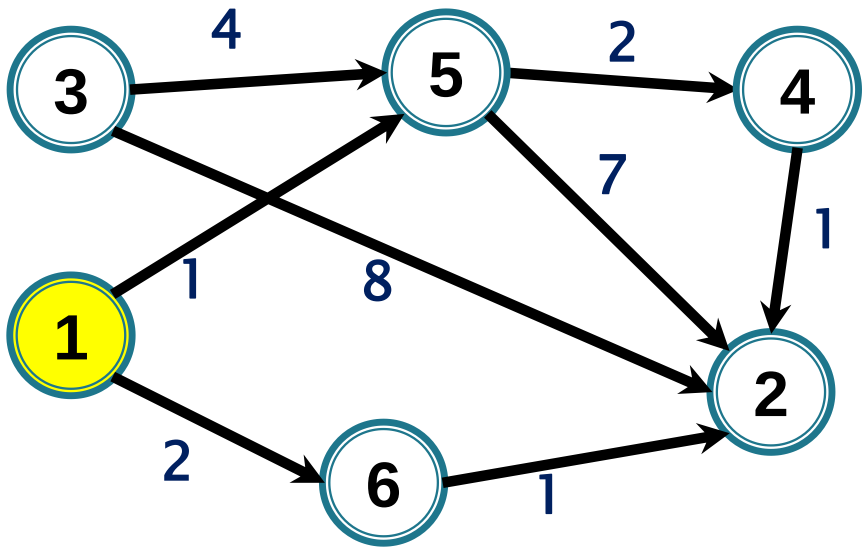
1, 3, 6, 5, 4, 2

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

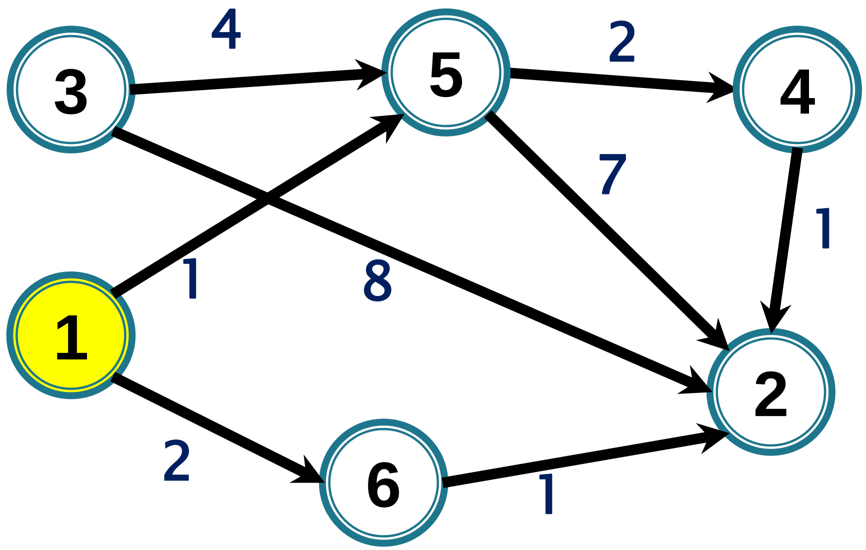
s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

d/tata	1	2	3	4	5	6
	[∞/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

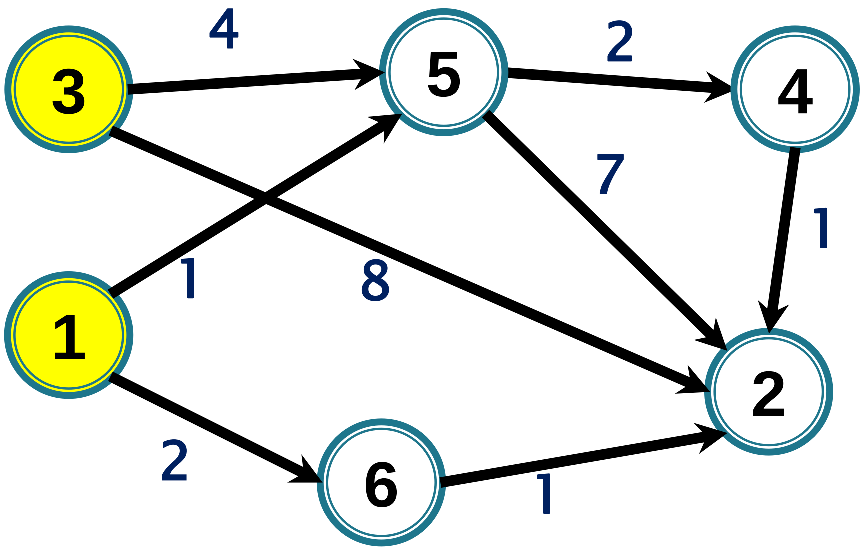
Ordine de calcul distanțe:

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

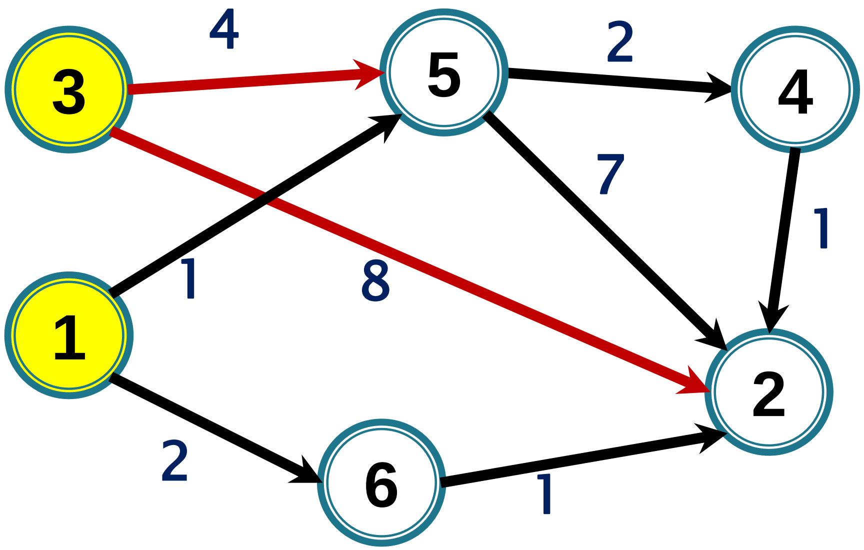
s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

	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

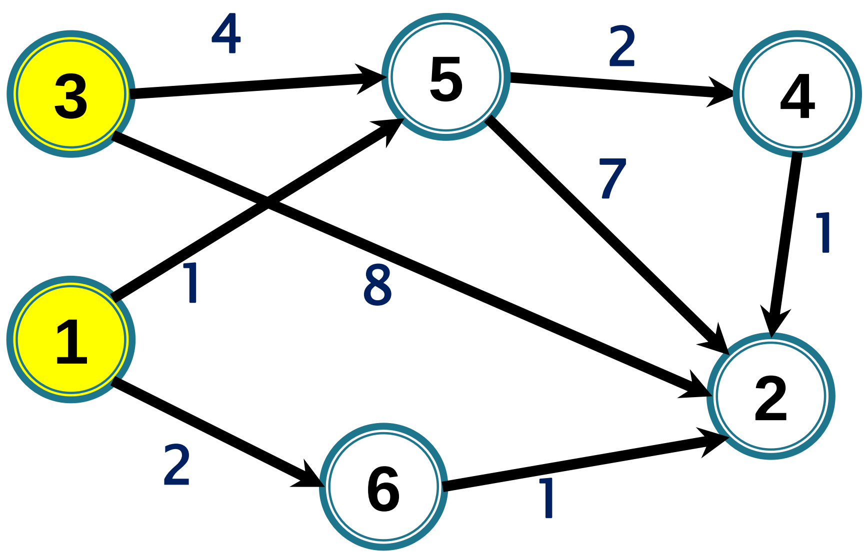
s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

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

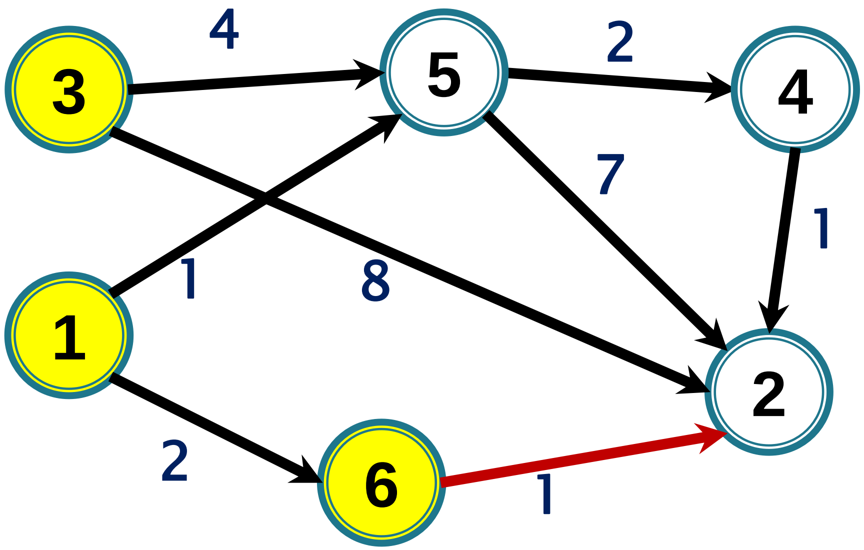
s=3 – vârf de start

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:	[∞/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

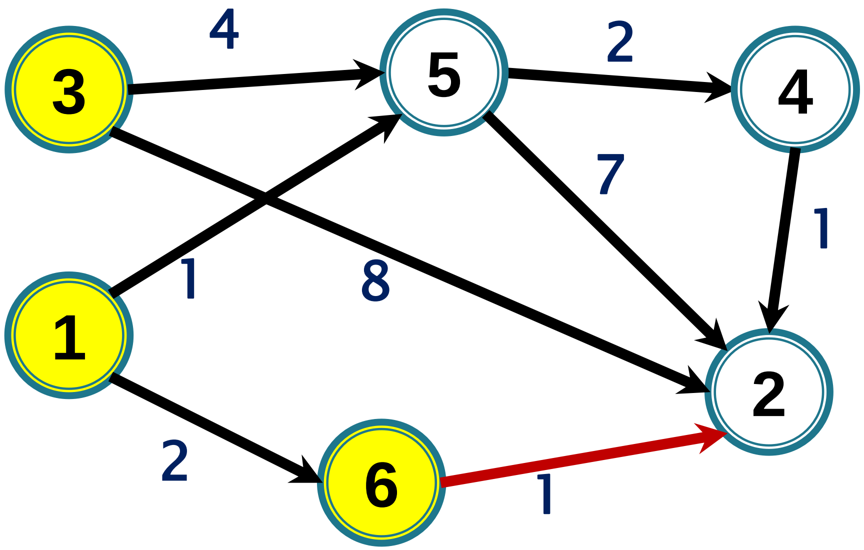
s=3 – vârf de start

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

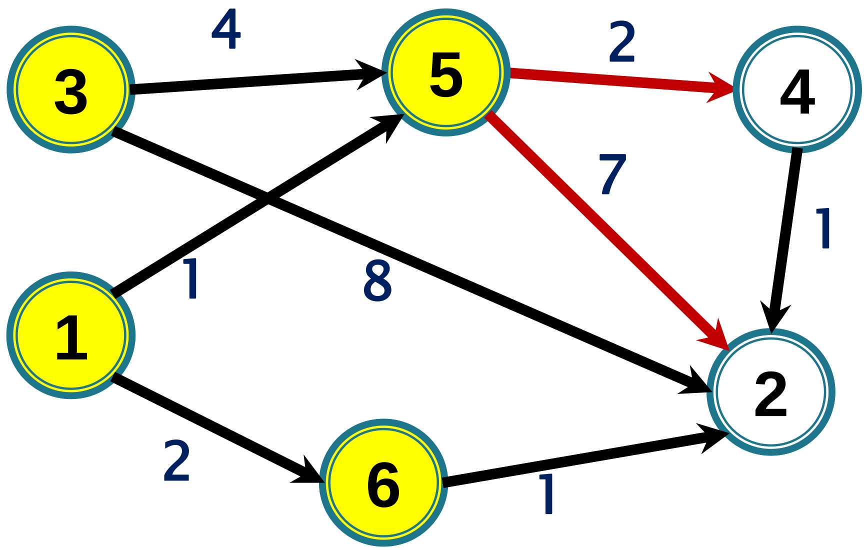
s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

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]

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



Sortare topologică

1, 3, 6, 5, 4, 2

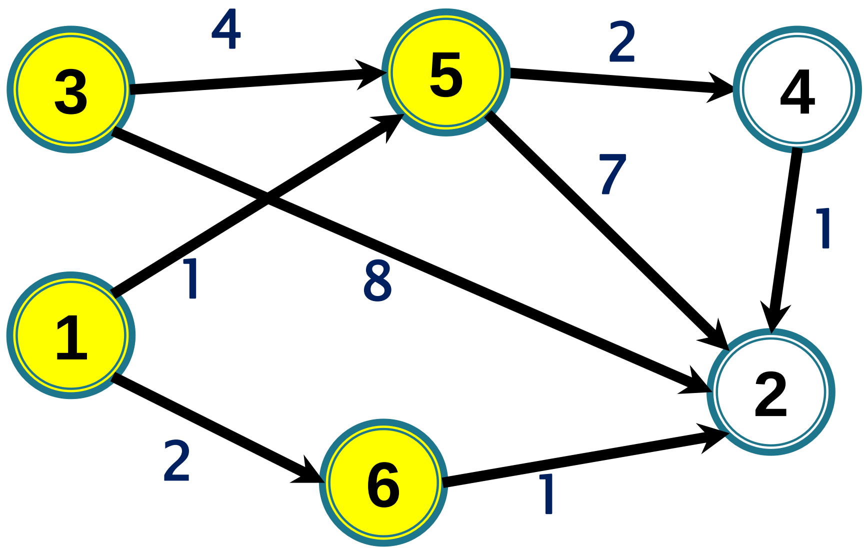
s=3 – vârf de start

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:						

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



Sortare topologică

1, 3, 6, 5, 4, 2

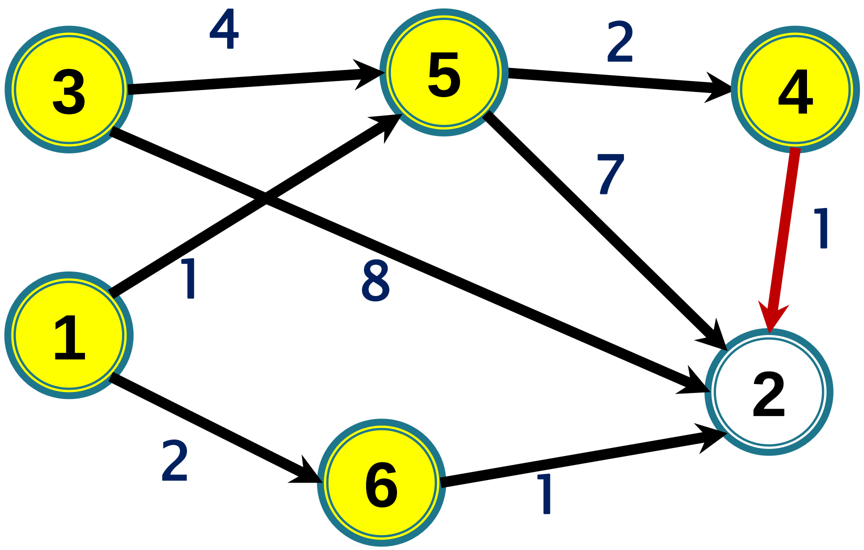
s=3 – vârf de start

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

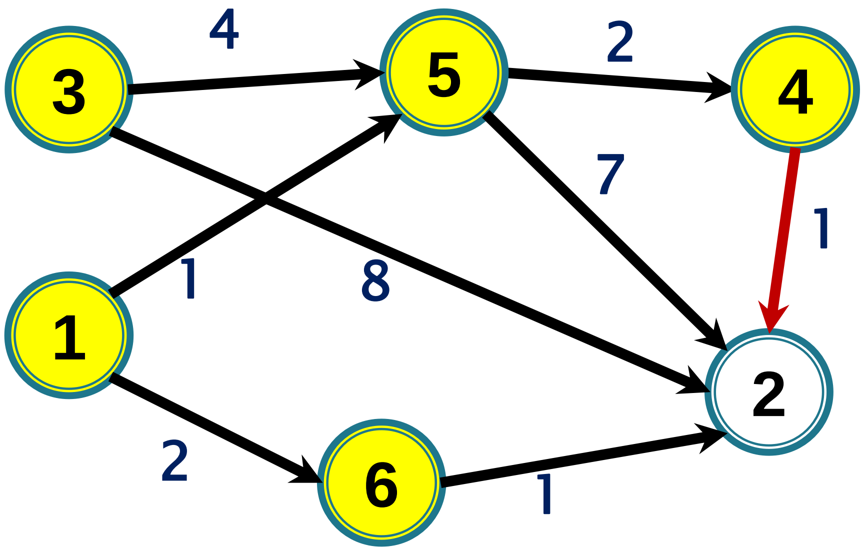
s=3 – vârf de start

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

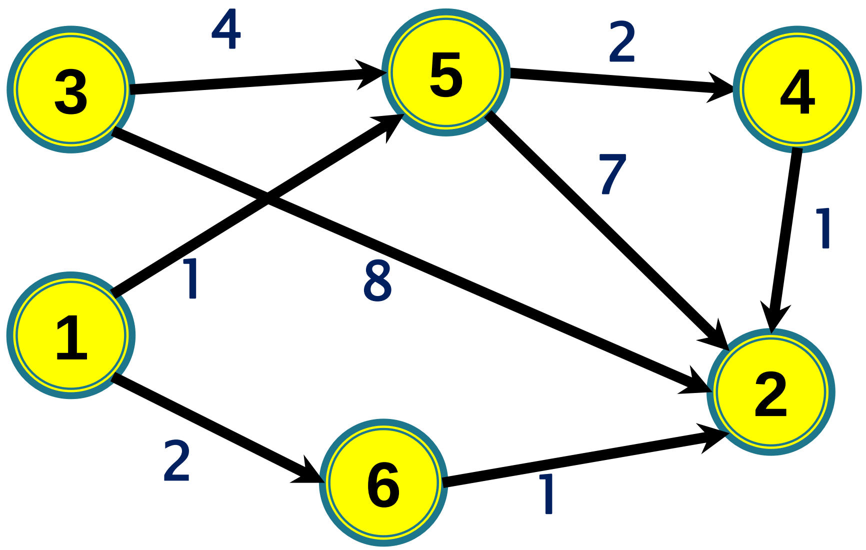
s=3 – vârf de start

Ordine de calcul distanțe:

1, 3, 6, 5, 4, 2

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]

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



Sortare topologică

1, 3, 6, 5, 4, 2

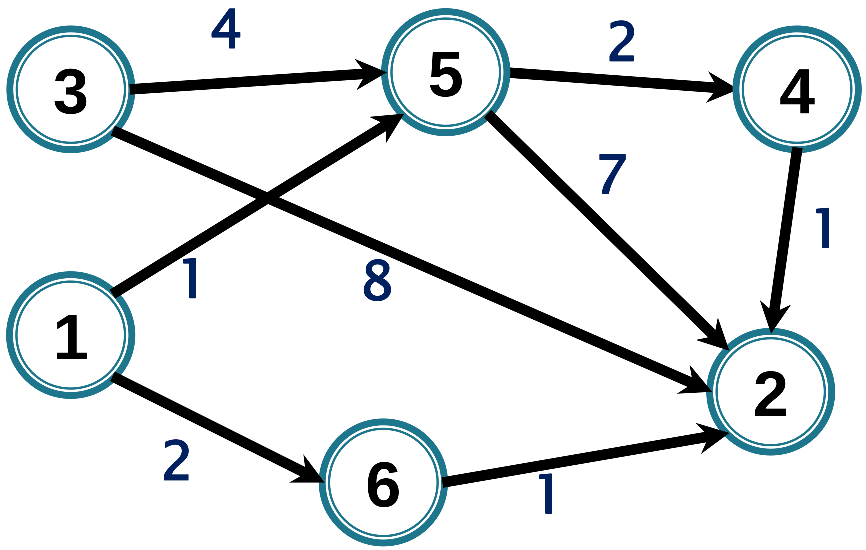
s=3 – vârf de start

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

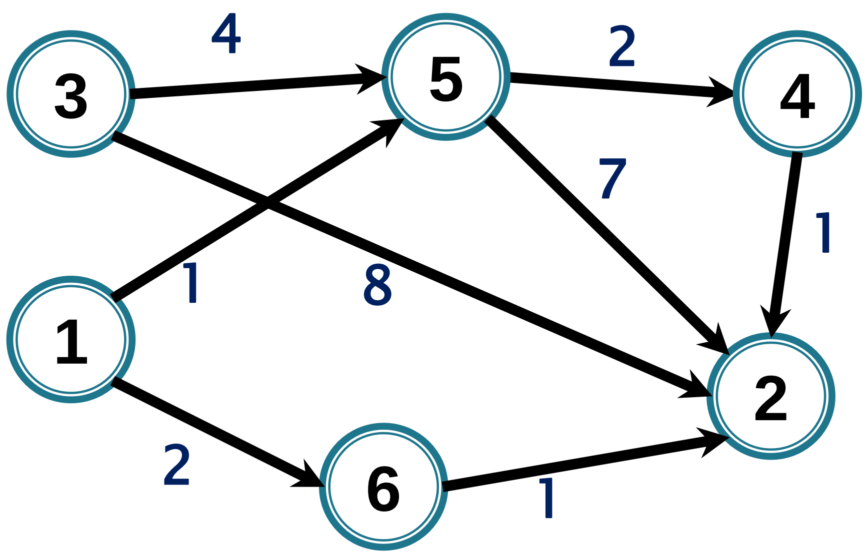
1, 3, 6, 5, 4, 2

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

d/tata	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

//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

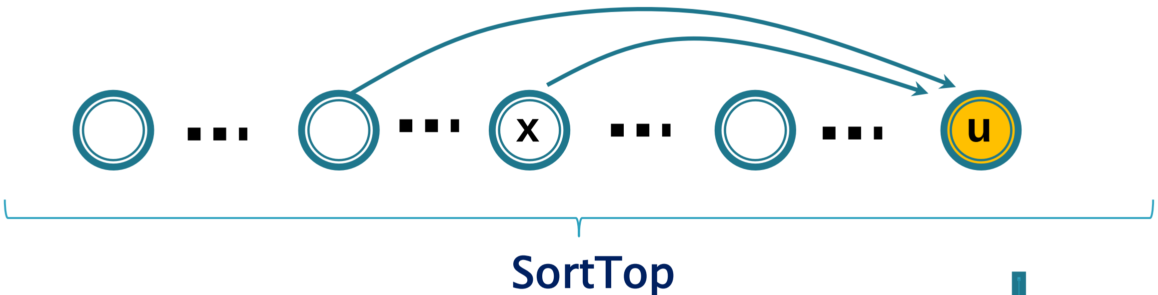
Drumuri minime de sursă unică în grafuri aciclice

- **Algoritmul funcționează corect și dacă există arce cu cost negativ**

Drumuri minime de sursă unică în grafuri aciclice

- **Algoritmul funcționează corect și dacă există arce cu cost negativ – Inducție după numărul de iterații**

Când algoritmul ajunge la vârful u avem



relaxare arce xu

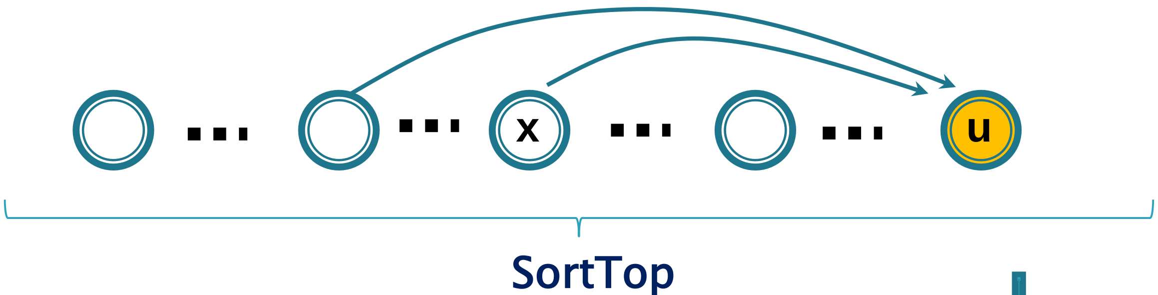
$$d[u] = \min\{\, d[x] + w(x,u) \mid xu \in E,\ \text{x înaintea lui u în SortTop} \,\}$$
$$= \min\{\, d[x] + w(x,u) \mid xu \in E \,\}$$

deja corect calculate (ipoteza inducție)

Drumuri minime de sursă unică în grafuri aciclice

- **Algoritmul funcționează corect și dacă există arce cu cost negativ – Inducție după numărul de iterații**

Când algoritmul ajunge la vârful u avem



relaxare arce xu

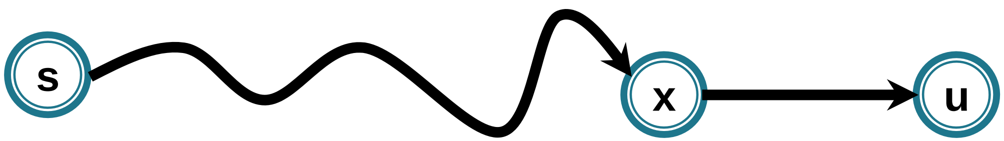
$$d[u] = \min\{\, d[x] + w(x,u) \mid xu \in E,\ \textcolor{red}{x \text{ înaintea lui } u \text{ în SortTop}} \,\}$$
$$= \min\{\, d[x] + w(x,u) \mid xu \in E \,\} = \min\{\delta(s, x) + w(x,u) \mid xu \in E \,\}$$
$$= \delta(s, u)$$

deja corect calculate (ipoteza inducție)

Drumuri minime de sursă unică în grafuri aciclice

Varianta 2 de demonstrație – similar Dijkstra

Fie P s-u drum minim și x predecesorul lui u pe acest drum.



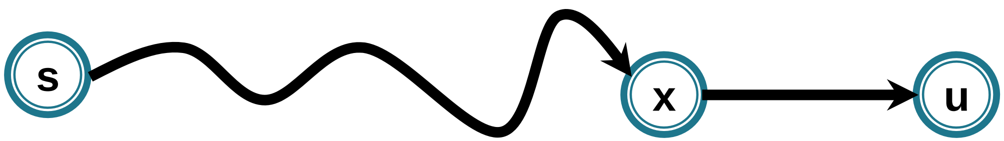
- x este înaintea lui u în SortTop => (ip. inducție)

  $$d[x] = \delta(s; x) = w([s \underline{P} x])$$

Drumuri minime de sursă unică în grafuri aciclice

Varianta 2 de demonstrație – similar Dijkstra

Fie P s-u drum minim și x predecesorul lui u pe acest drum.



- x este înaintea lui u în SortTop => (ip. inducție)

$$d[x] = \delta(s; x) = w([s \underline{P} x])$$

- după relaxarea arcului xu avem:

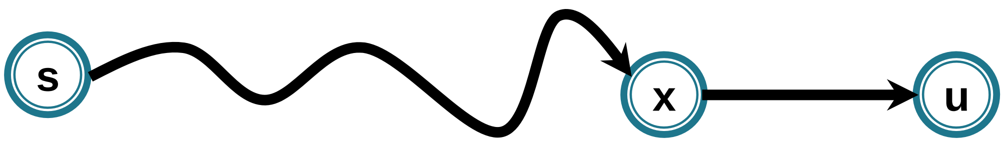
$$d[u] \le d[x] + w(xu) = w([s \underline{P} x]) + w(xu) =$$

$$= w([s \underline{P} u]) = \delta(s; u)$$

Drumuri minime de sursă unică în grafuri aciclice

Varianta 2 de demonstrație – similar Dijkstra

Fie P s-u drum minim și x predecesorul lui u pe acest drum.



- x este înaintea lui u în SortTop => (ip. inducție)

$$d[x] = \delta(s; x) = w([s \underline{P} x])$$

- după relaxarea arcului xu avem:

$$d[u] \le d[x] + w(xu) = w([s \underline{P} x]) + w(xu) =$$
$$= w([s \underline{P} u]) = \delta(s; u)$$

Dar $\delta(s; u) \le d[u]$ (estimare superioară) => $\delta(s; u) = d[u]$

Aplicație –
Drumuri critice

Drumuri critice

- **Se cunosc pentru un proiect cu n activități, numerotate 1,..., n:**
  - **durata** fiecărei activități
  -
  -

Drumuri critice

- **Se cunosc pentru un proiect cu n activități, numerotate 1,..., n:**
  - **durata** fiecărei activități
  - perechi (i, j) = activitatea i trebuie să se încheie **înainte** să înceapă j (activitatea j depinde de i)
  -

Drumuri critice

- **Se cunosc pentru un proiect cu n activități, numerotate 1,..., n:**
  - **durata** fiecărei activități
  - perechi (i, j) = activitatea i trebuie să se încheie **înainte** să înceapă j
  - activitățile se pot desfășura **și în paralel**

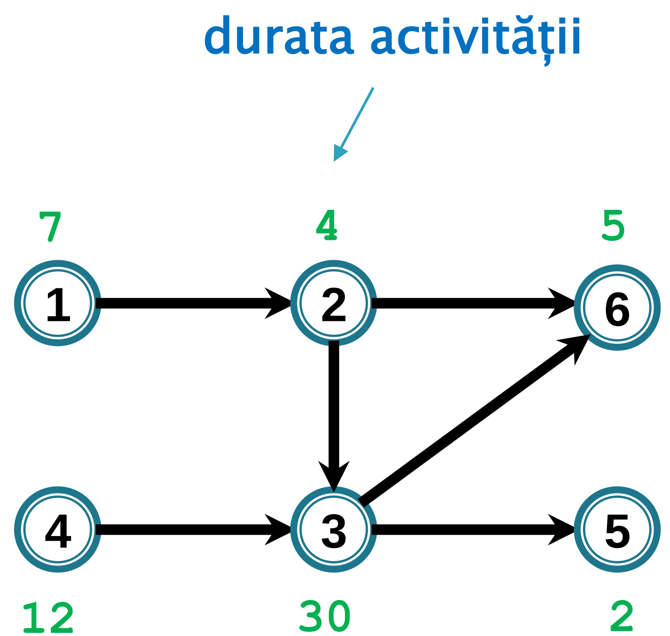
Drumuri critice

- **Se cunosc pentru un proiect cu n activități, numerotate 1,..., n:**
  - **durata** fiecărei activități
  - perechi (i, j) = activitatea i trebuie să se încheie **înainte** să înceapă j
  - activitățile se pot desfășura și **în paralel**

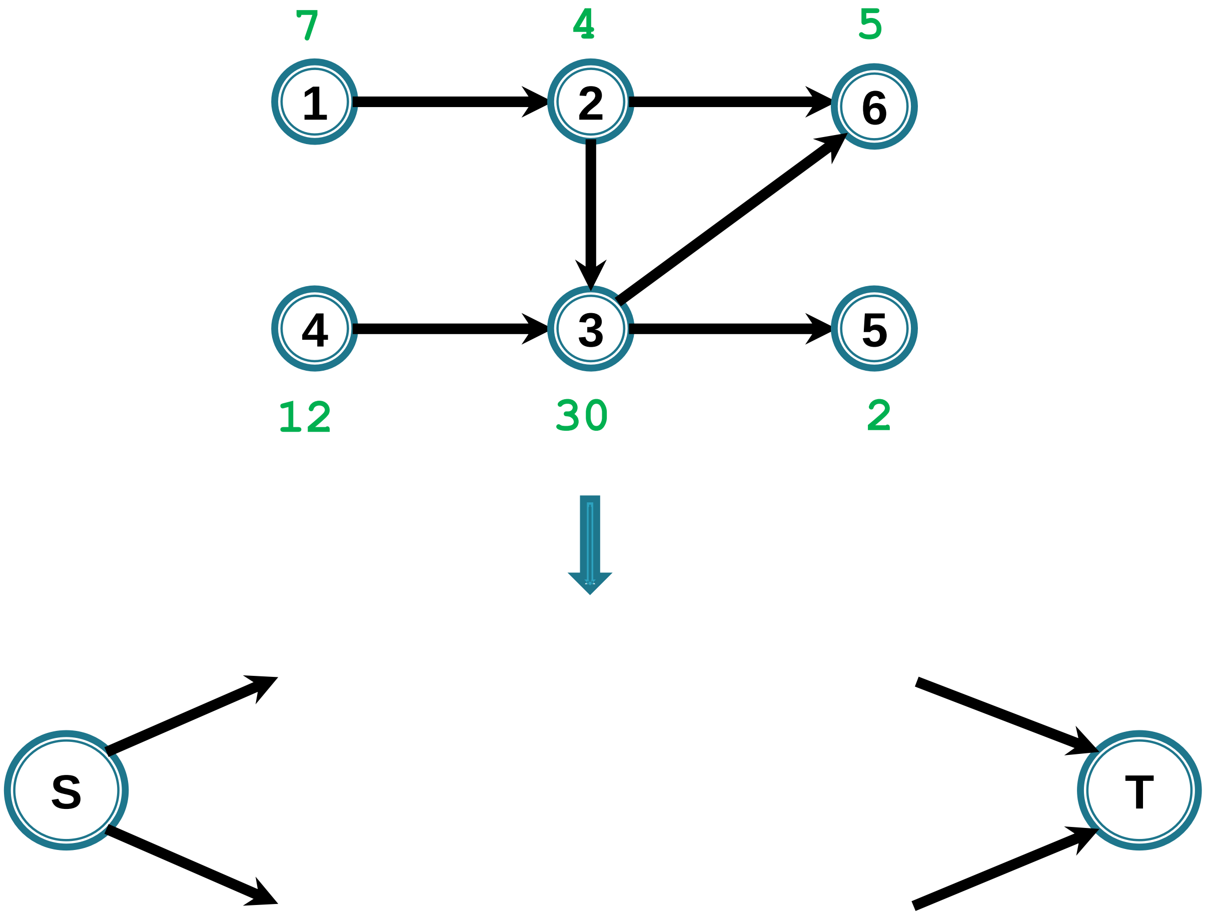
Se cere: **timpul minim de finalizare a proiectului** (dacă începe la ora 0) + **planificarea activităților**

Drumuri critice

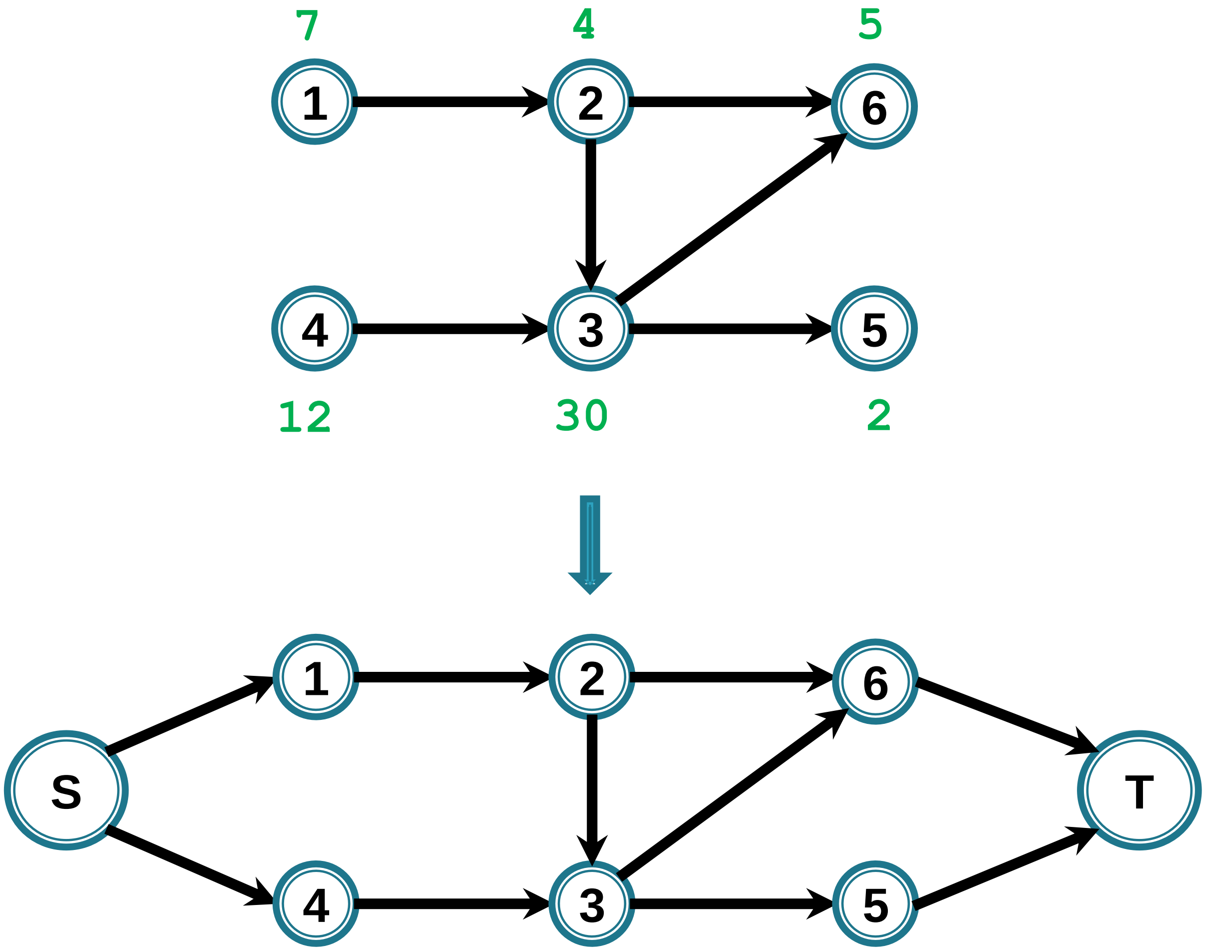
- n = 6
  - Activitatea 1 – durata 7
  - Activitatea 2 – durata 4
  - Activitatea 3 – durata 30
  - Activitatea 4 – durata 12
  - Activitatea 5 – durata 2
  - Activitatea 6 – durata 5
  - (1, 2)
  - (2, 3)
  - (3, 6)
  - (4, 3)
  - (2, 6)
  - (3, 5)



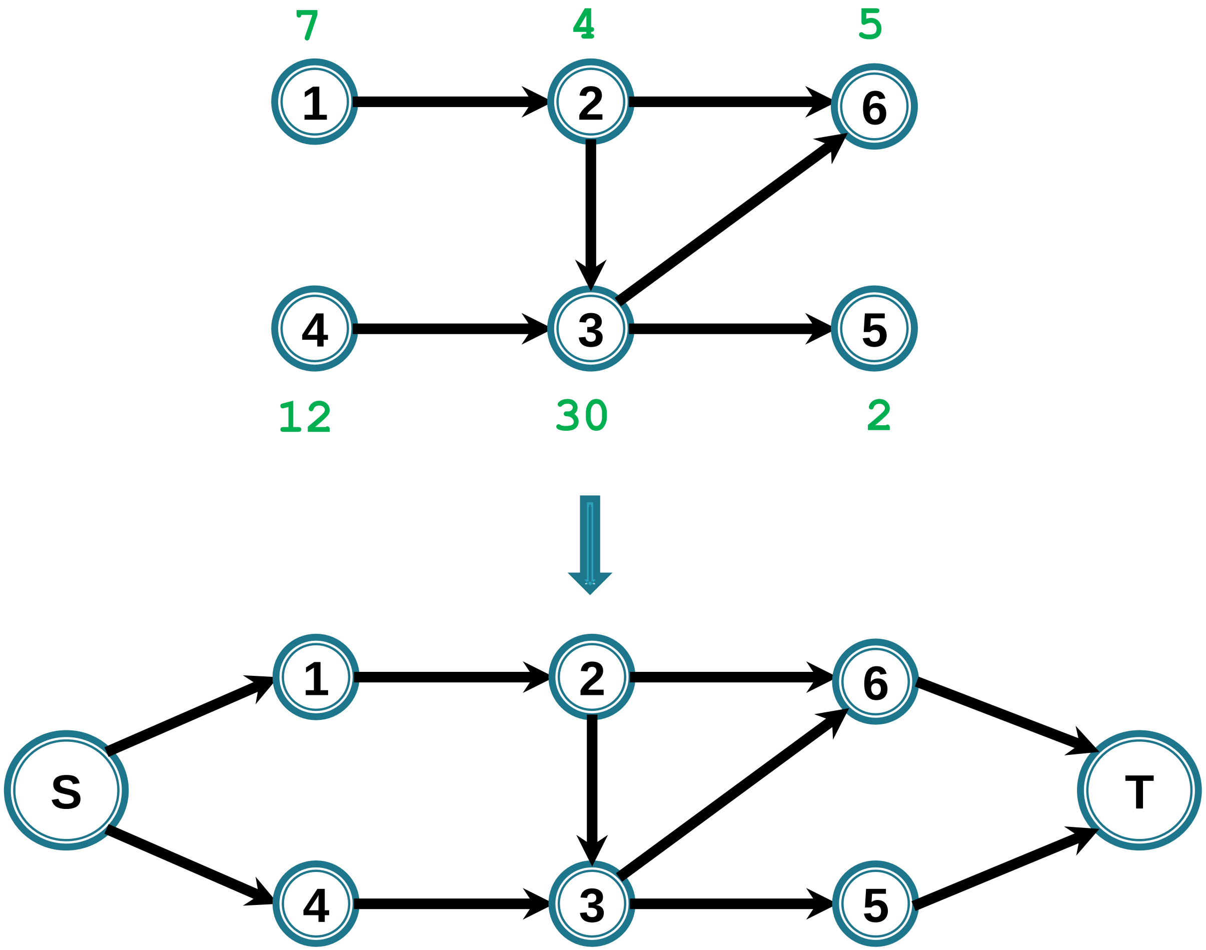
Drumuri critice



Drumuri critice

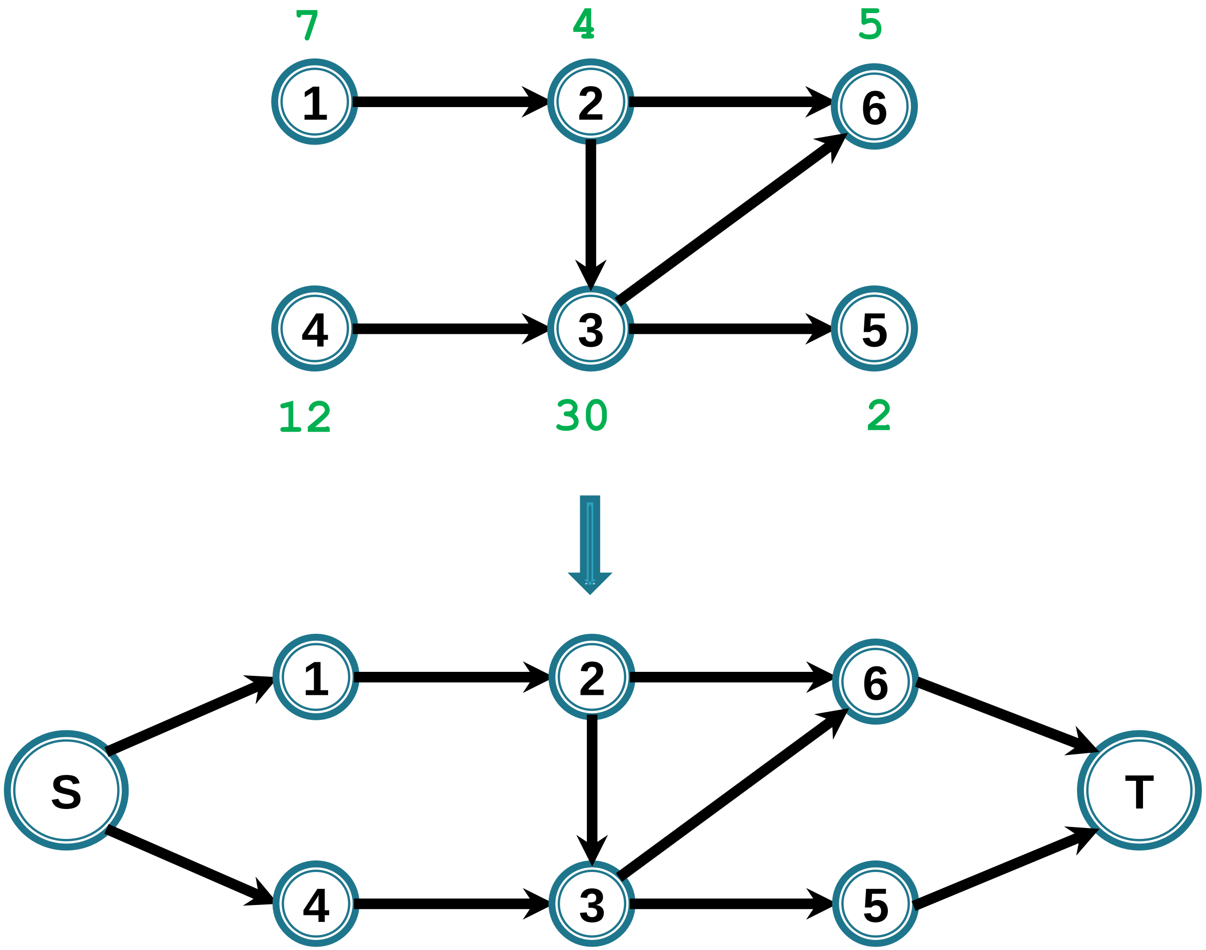


Drumuri critice



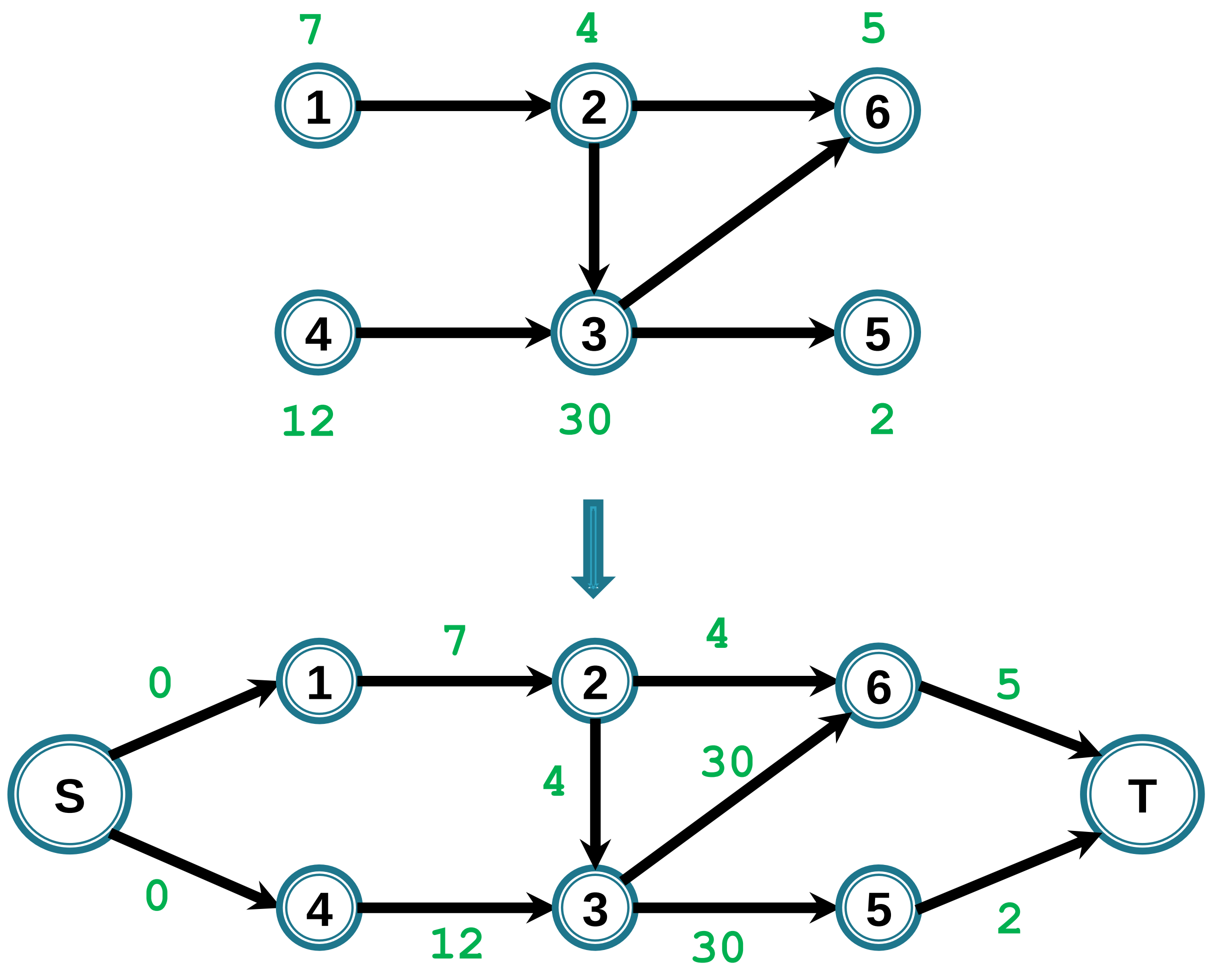
w(i,j) = ?

Drumuri critice



w(i,j) = durata activității i
= întârzierea minimă între începutul activității i și începutul activității j (mai general)

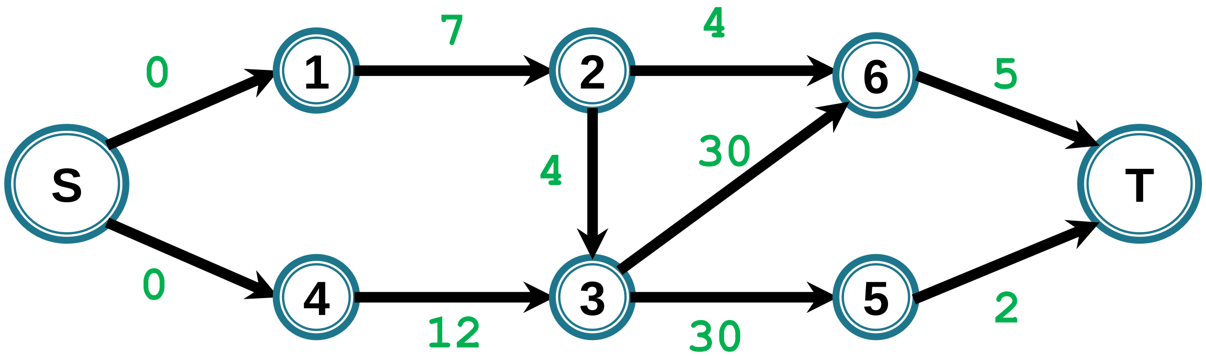
Drumuri critice



w(i,j) = durata activității i

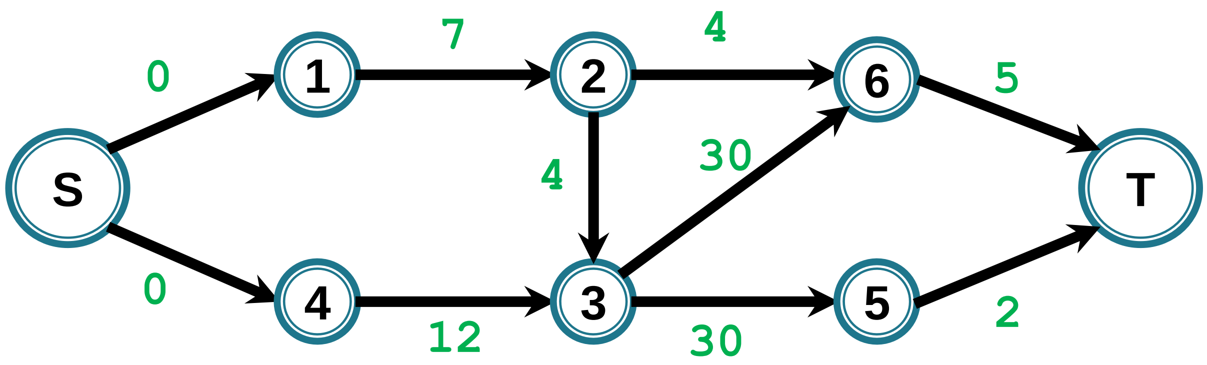
= întârzierea minimă între începutul activității i și începutul activității j (mai general)

Drumuri critice



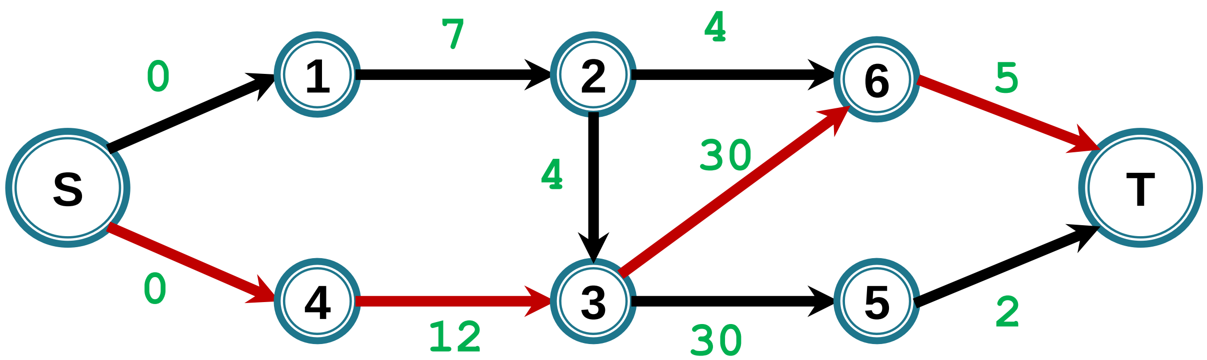
Timpul minim de finalizare a proiectului = ?

Drumuri critice

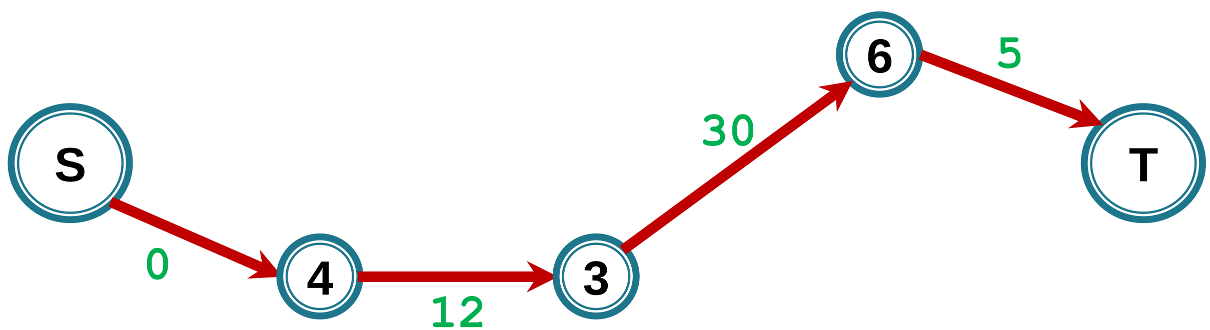


Timpul minim de finalizare a proiectului = costul maxim al unui drum de la S la T

Drumuri critice



Timpul minim de finalizare a proiectului = **costul maxim al unui drum de la S la T**



Drum CRITIC

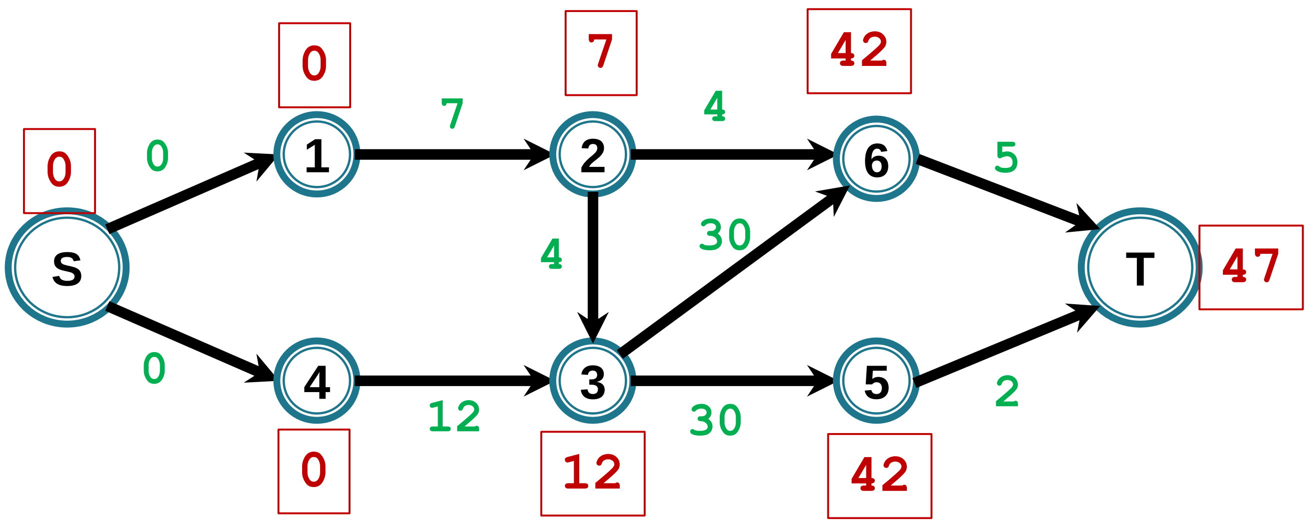
Drumuri critice

- Durata minimă a proiectului = costul maxim al unui drum de la S la T
  - Drum critic = drum de cost maxim de la S la T
  - Orice întârziere în desfăşurarea unei activităţi de pe acest drum duce la creşterea timpului de terminare al proiectului
  - PERT/CPM – Program Evaluation and Review Technique / Critical Path Method

Drumuri critice

- **Durata minimă a proiectului = costul maxim al unui drum de la S la T**

- **Timpul minim de început al unei activități u =** costul maxim al unui drum de la S la u



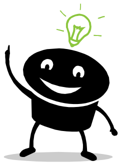
activitatea 1: intervalul de desfășurare (0,7)

activitatea 3: intervalul de desfășurare (12, 42)

Drumuri critice

Putem modifica algoritmul de determinare de drumuri minime în grafuri aciclice a.î. să determine drumuri maxime (de cost maxim) de la s la celelalte vârfuri?

Drumuri critice



Putem modifica algoritmul de determinare de drumuri minime în grafuri aciclice a.î. să determine drumuri maxime (de cost maxim) de la S la celelalte vârfuri

- Problema este echivalentă cu a determina **drumuri minime** din S în graful în care **înlocuim fiecare pondere w(e) cu –w(e)**

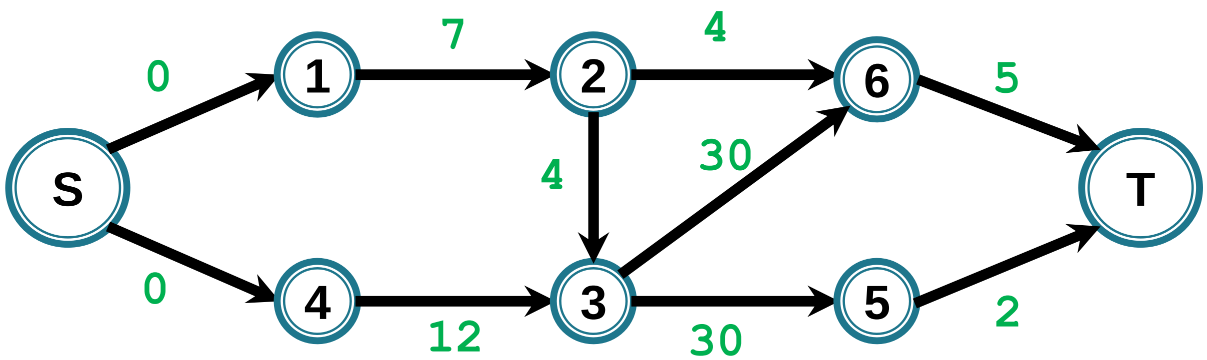
- Modificăm astfel doar inițializarea distanțelor (cu $-\infty$ în loc de $+\infty$) și **inversam condiția de la relaxarea** arcelor pentru a calcula maxim în loc de minim

- **Corectitudine** – rezultă din corectitudinea algoritmului pentru drumul minim

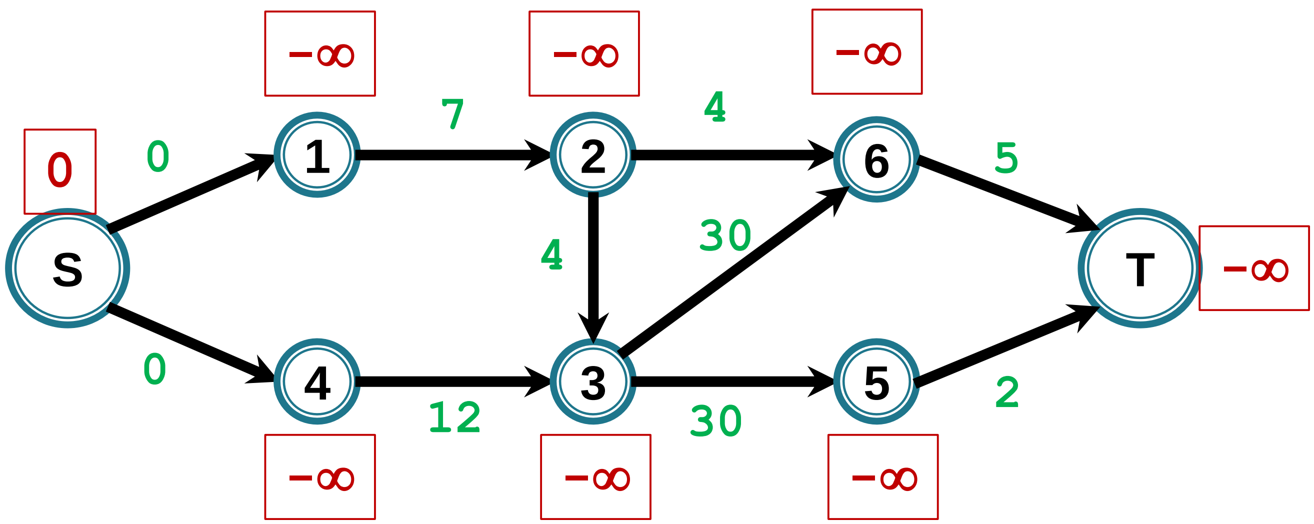
Drumuri maxime de sursă unică în grafuri aciclice

```
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
```

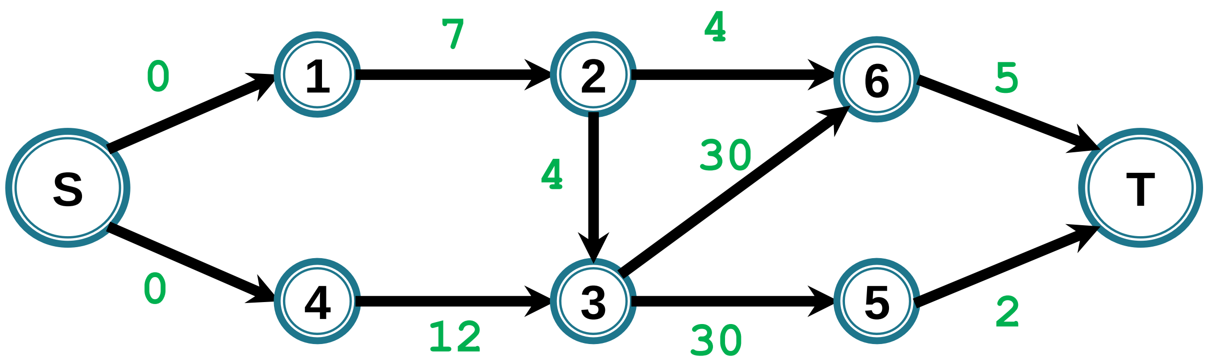
Drumuri critice



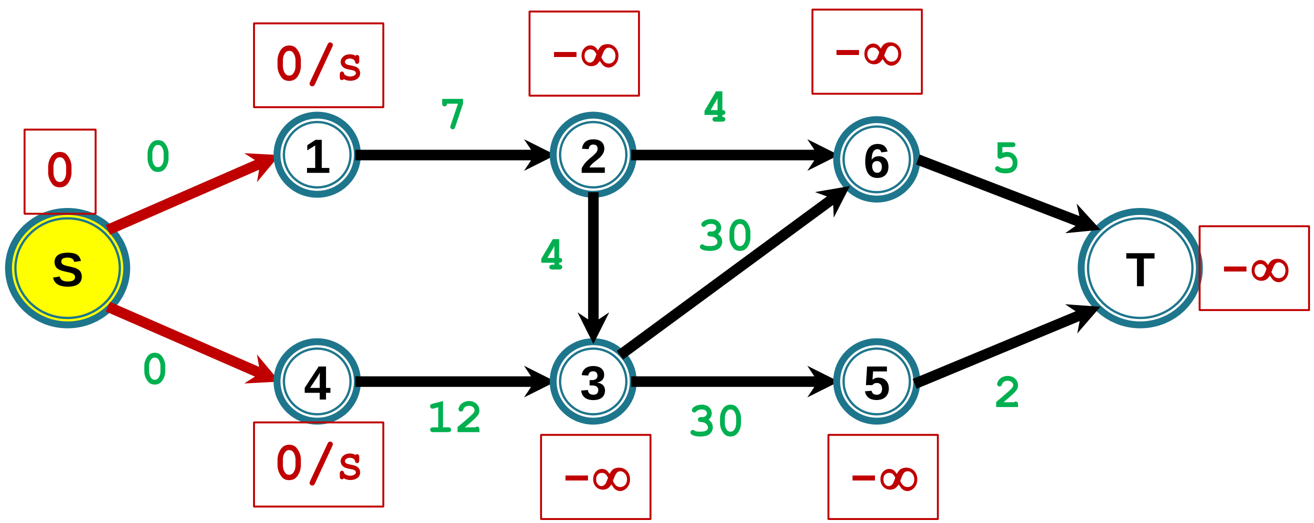
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



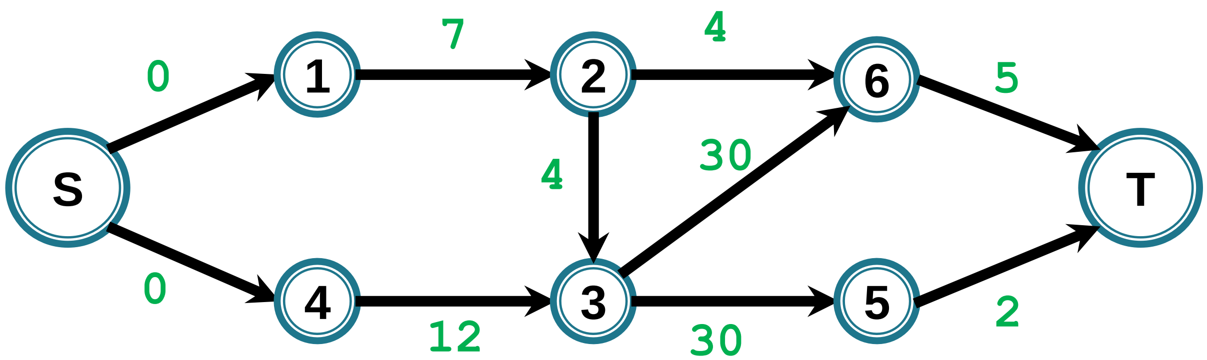
Drumuri critice



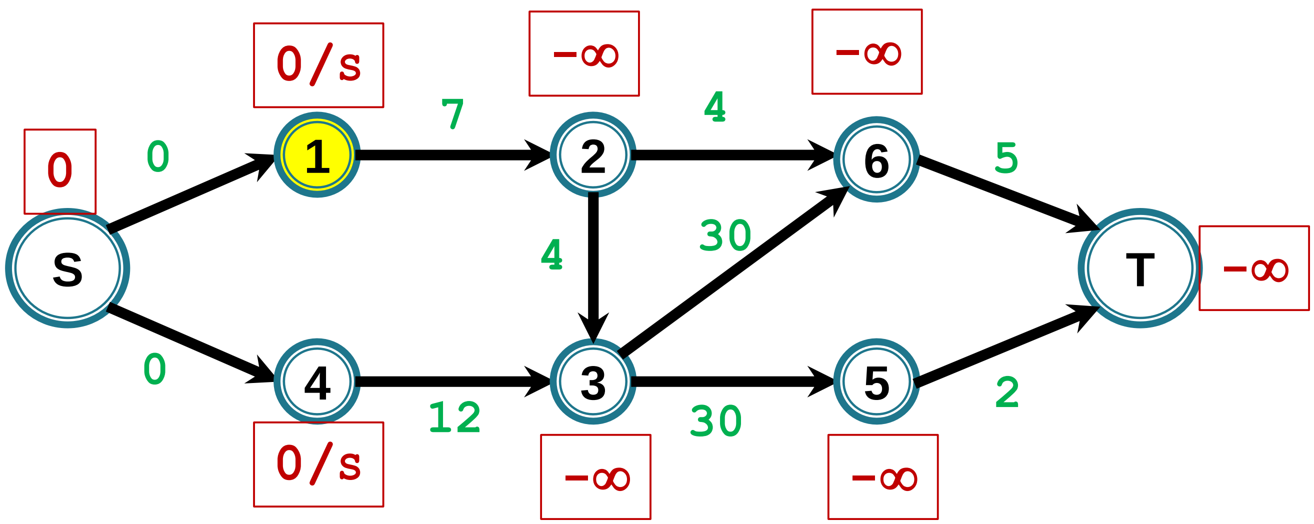
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



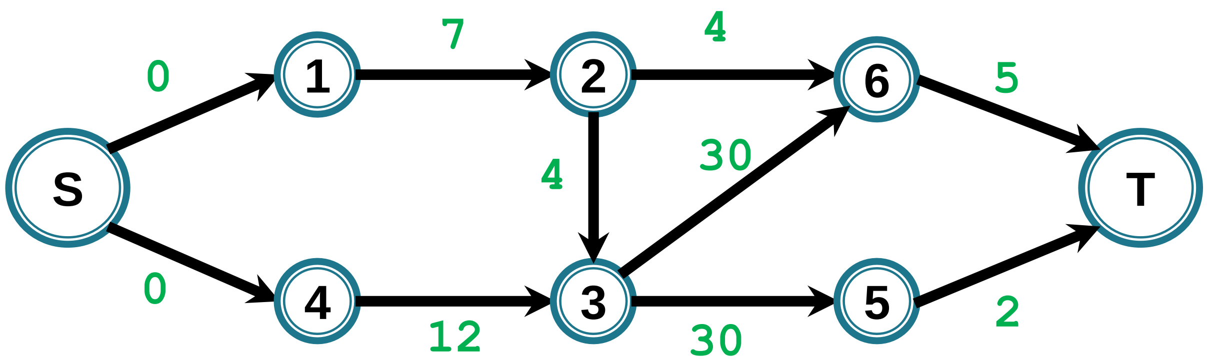
Drumuri critice



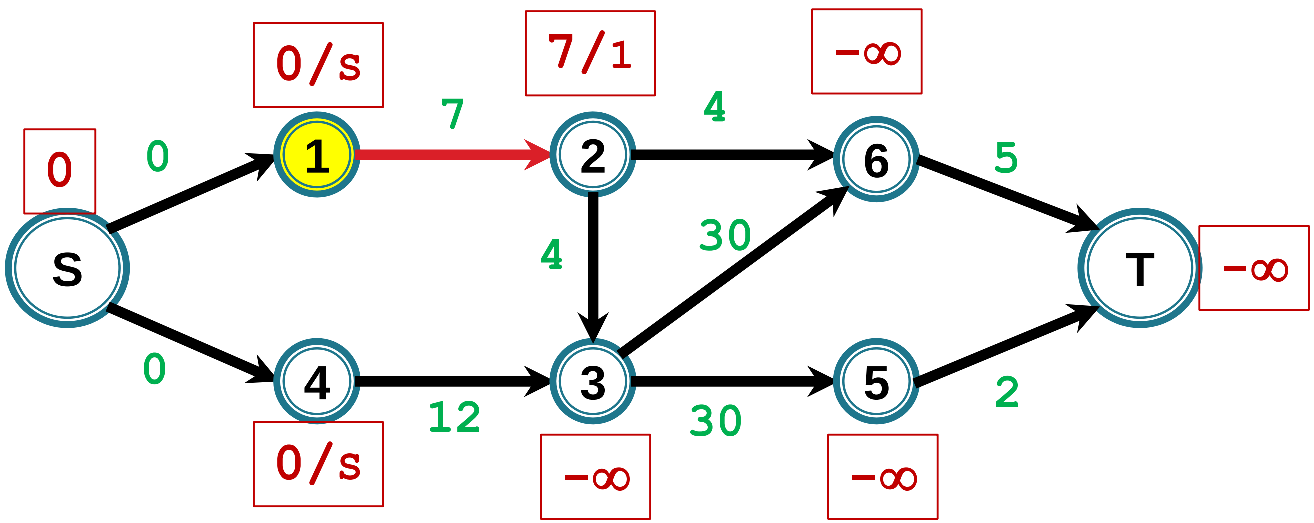
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



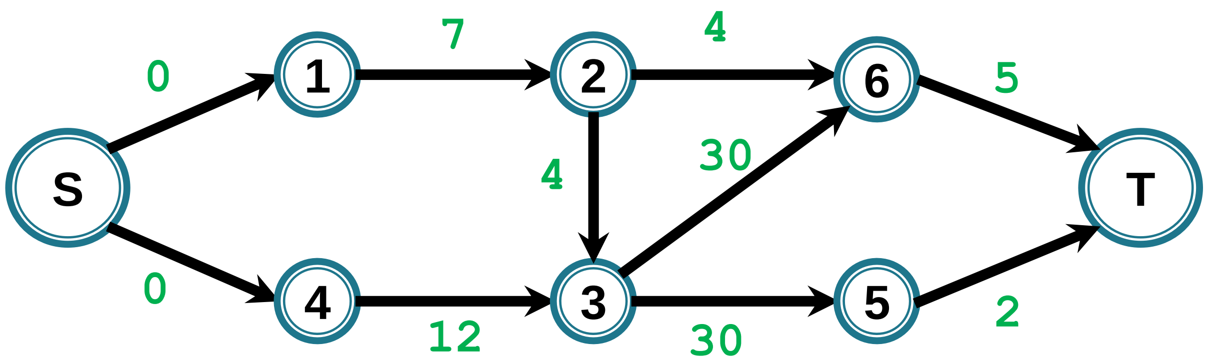
Drumuri critice



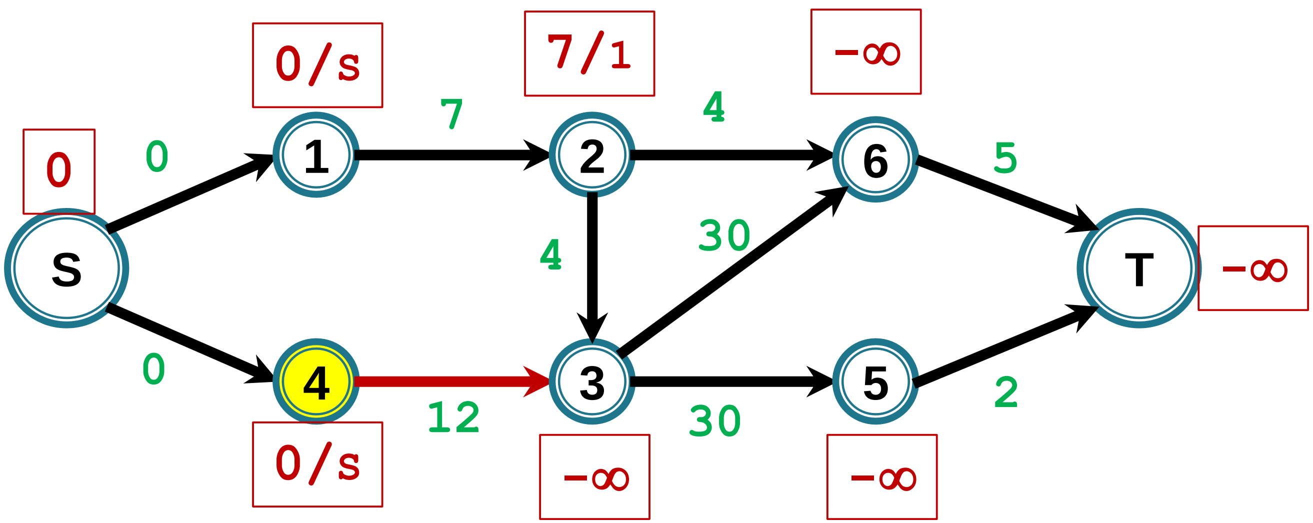
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



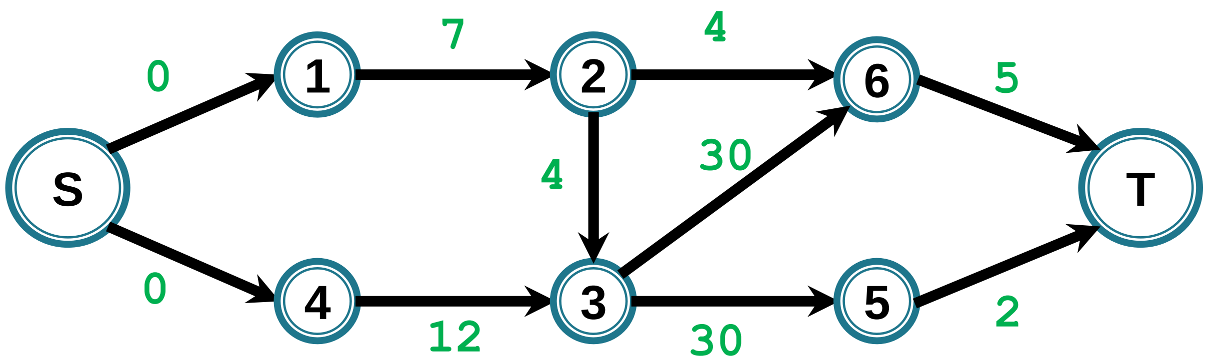
Drumuri critice



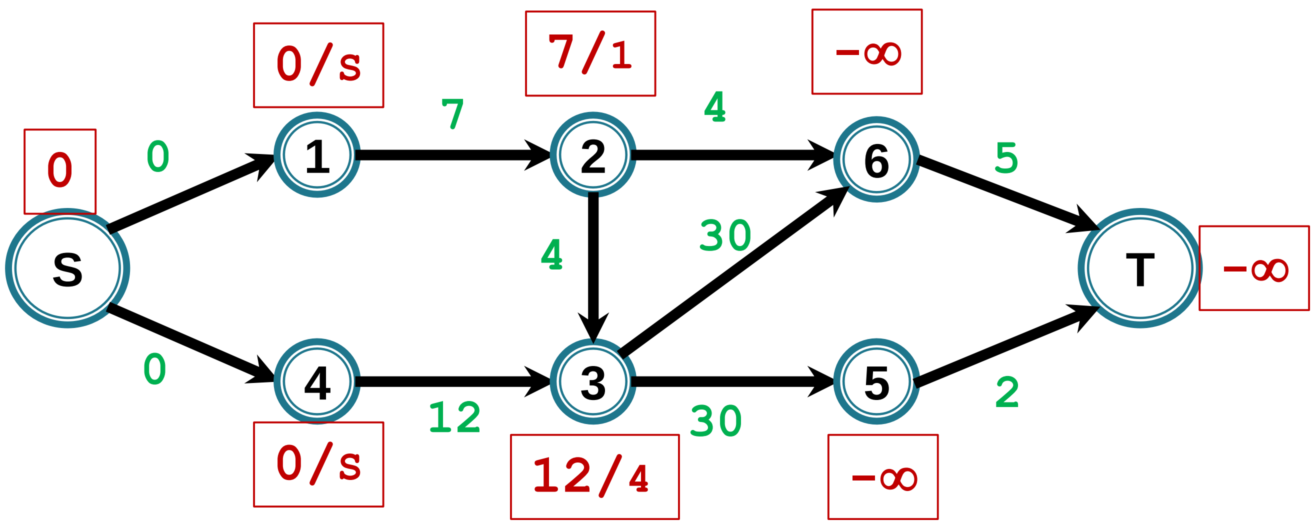
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



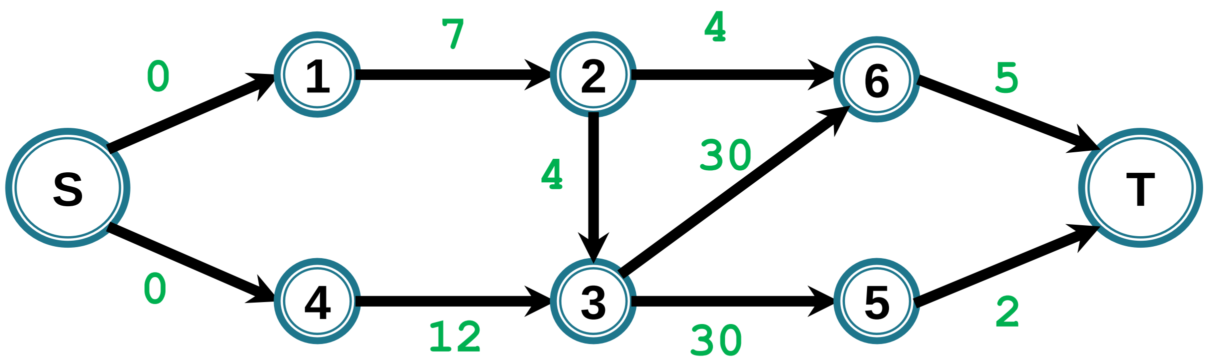
Drumuri critice



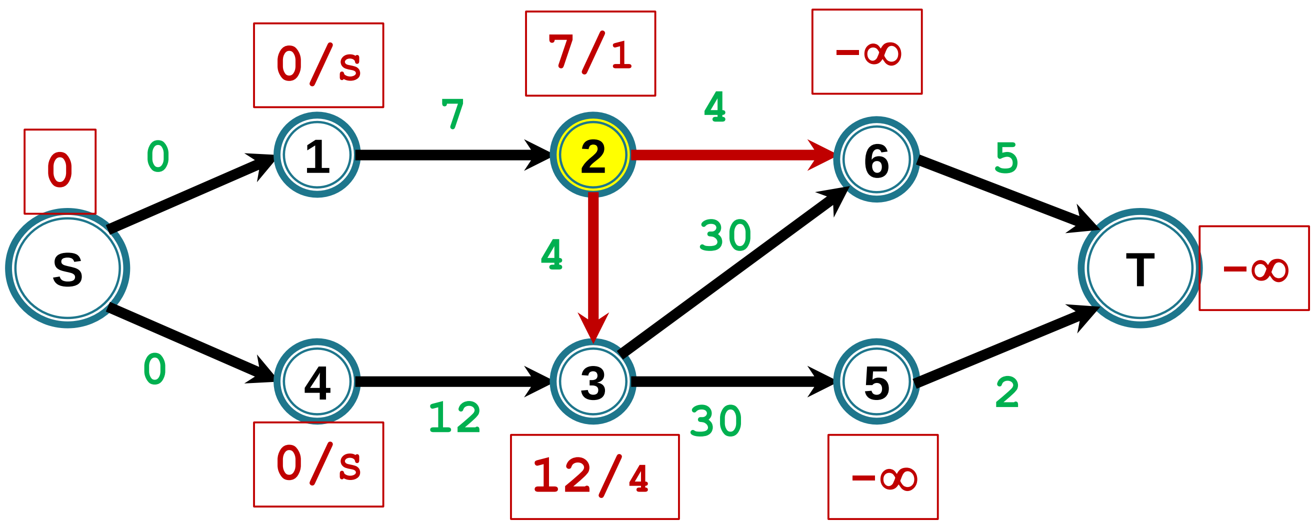
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



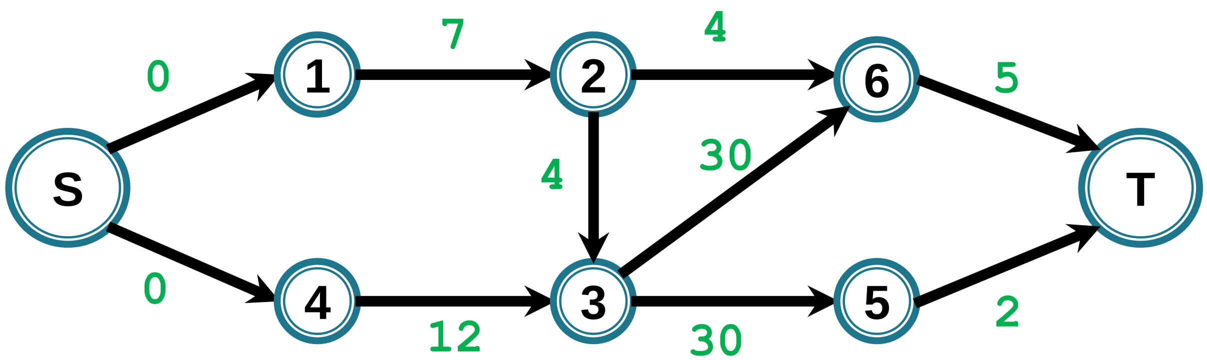
Drumuri critice



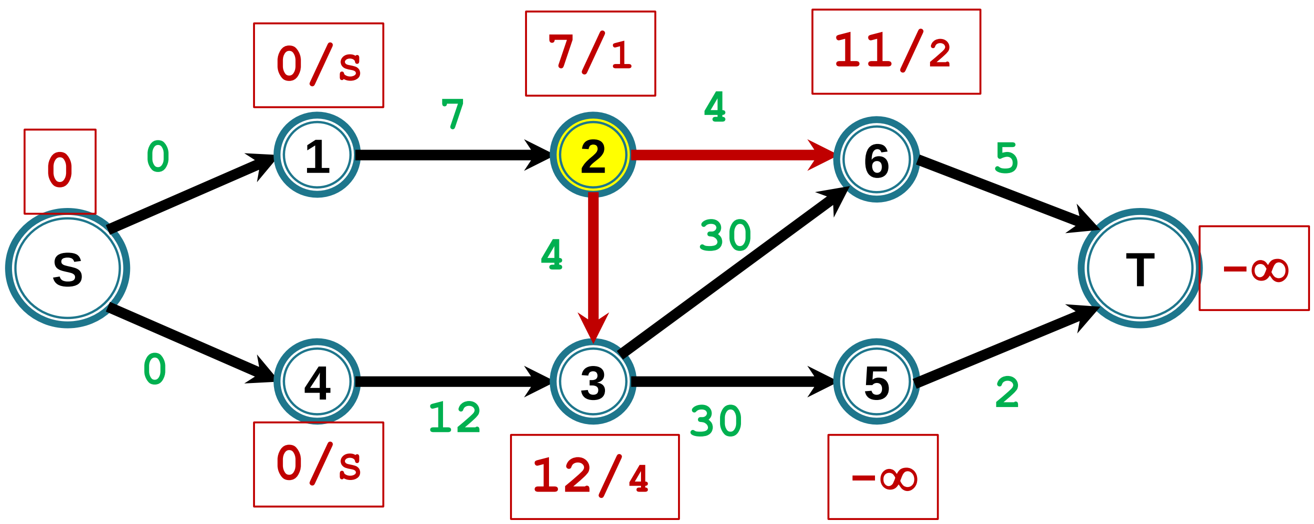
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



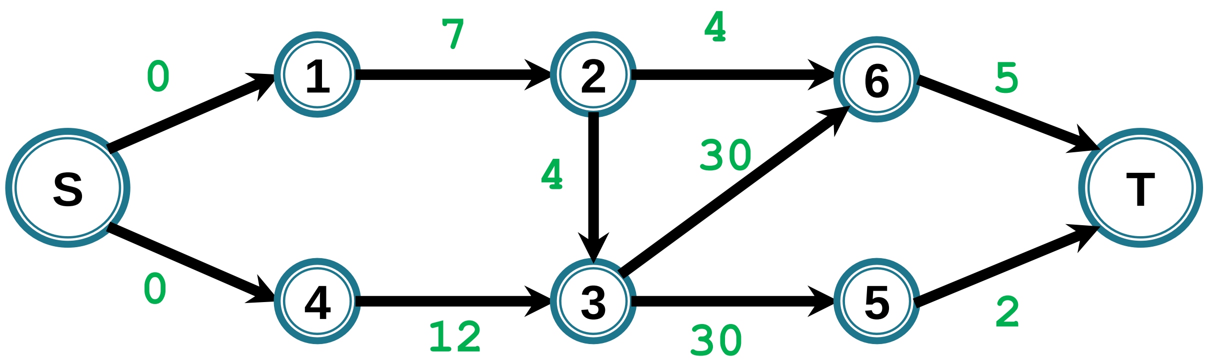
Drumuri critice



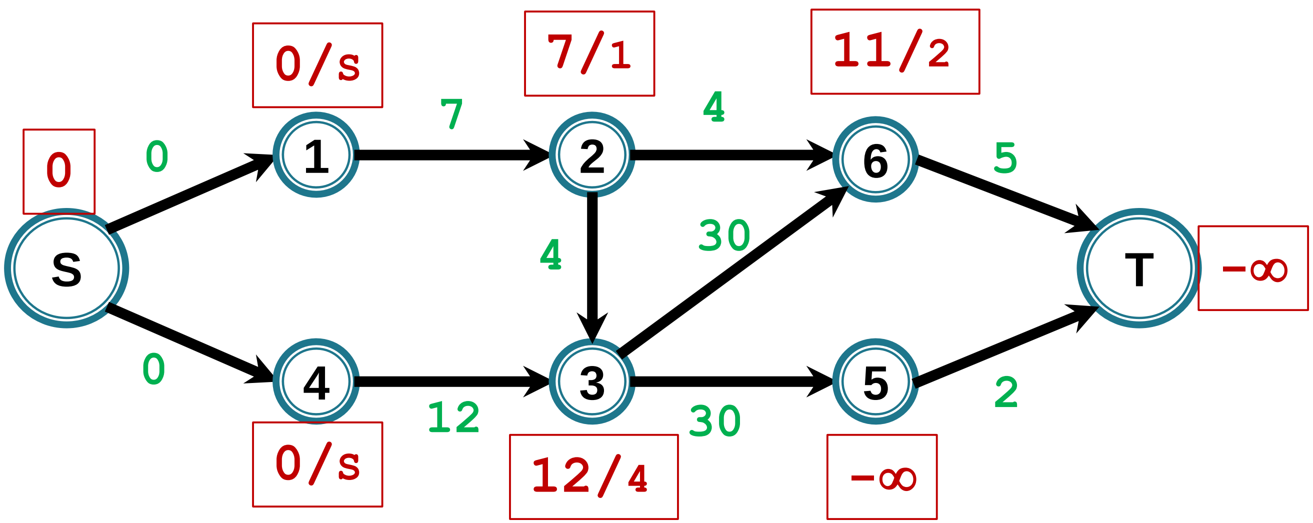
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



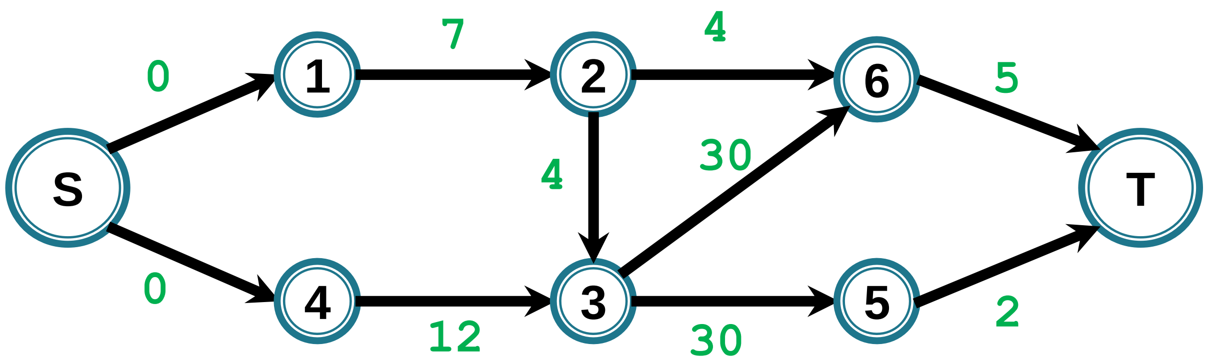
Drumuri critice



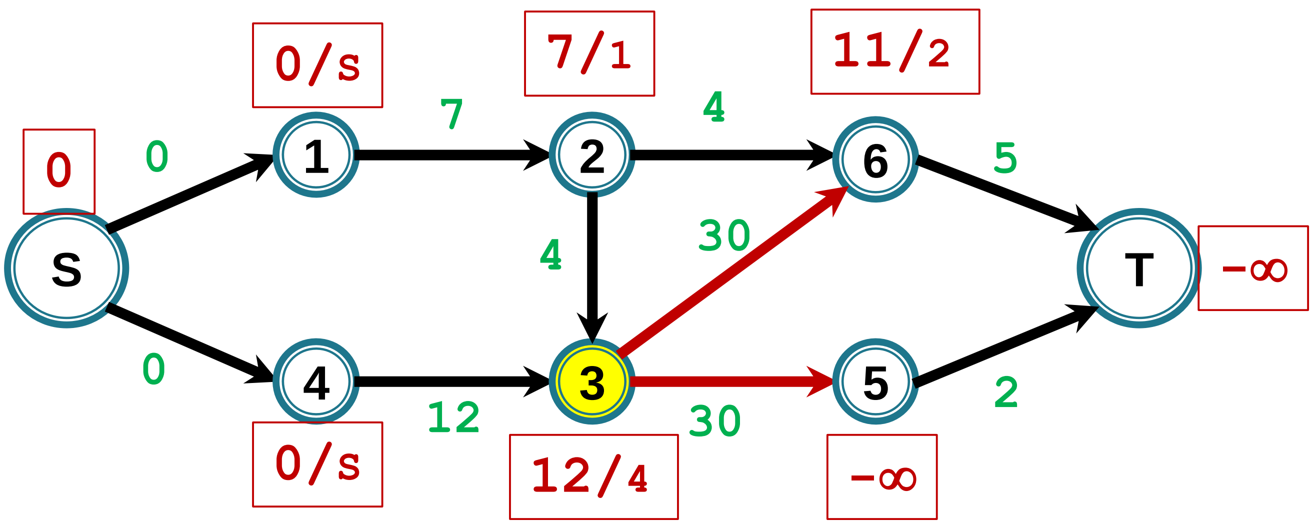
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



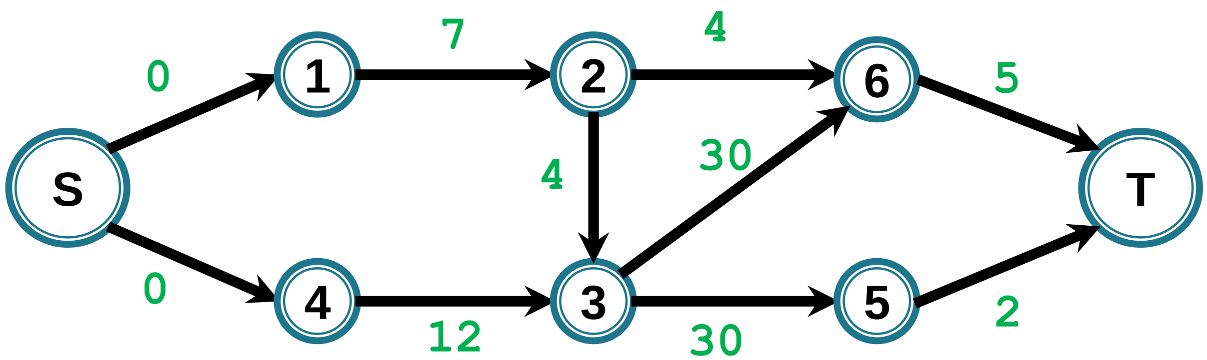
Drumuri critice



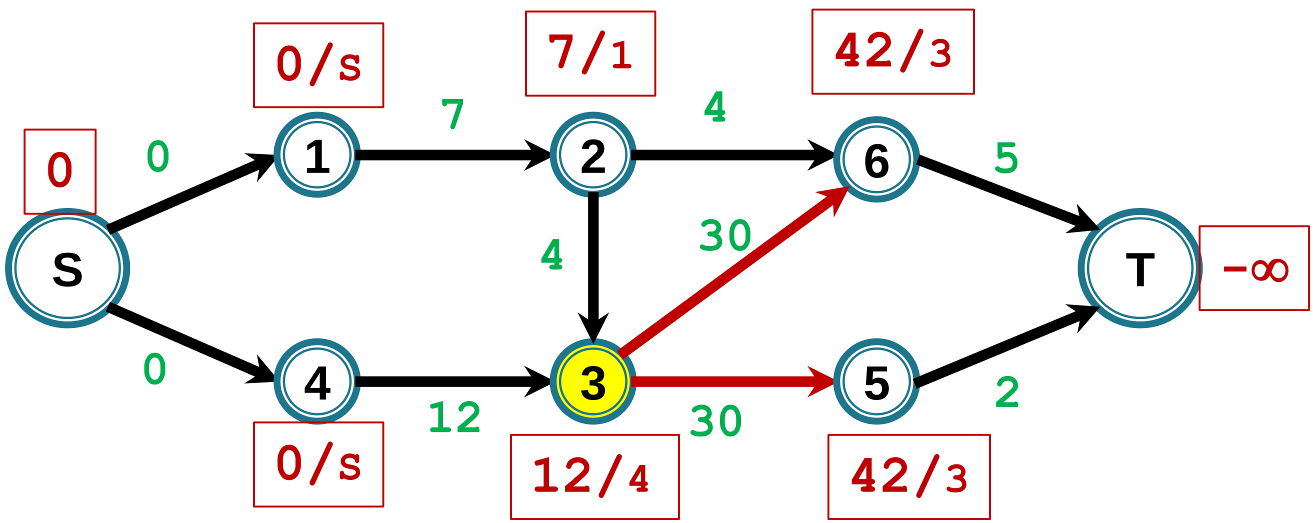
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



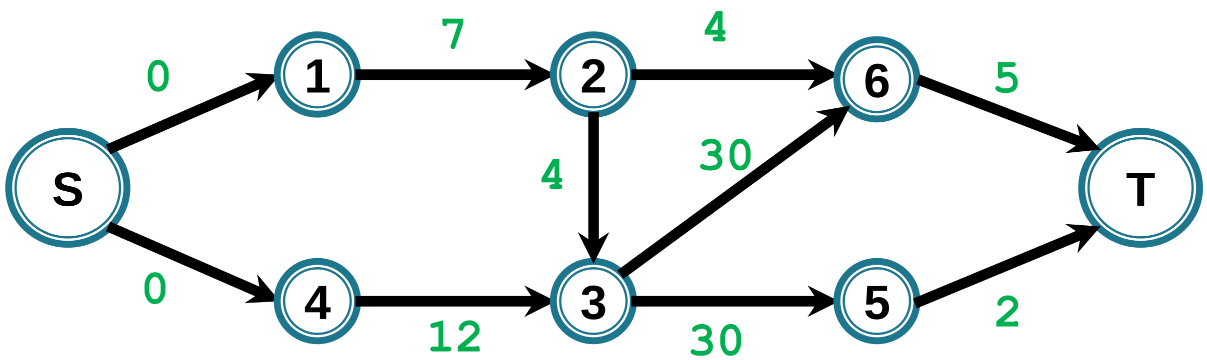
Drumuri critice



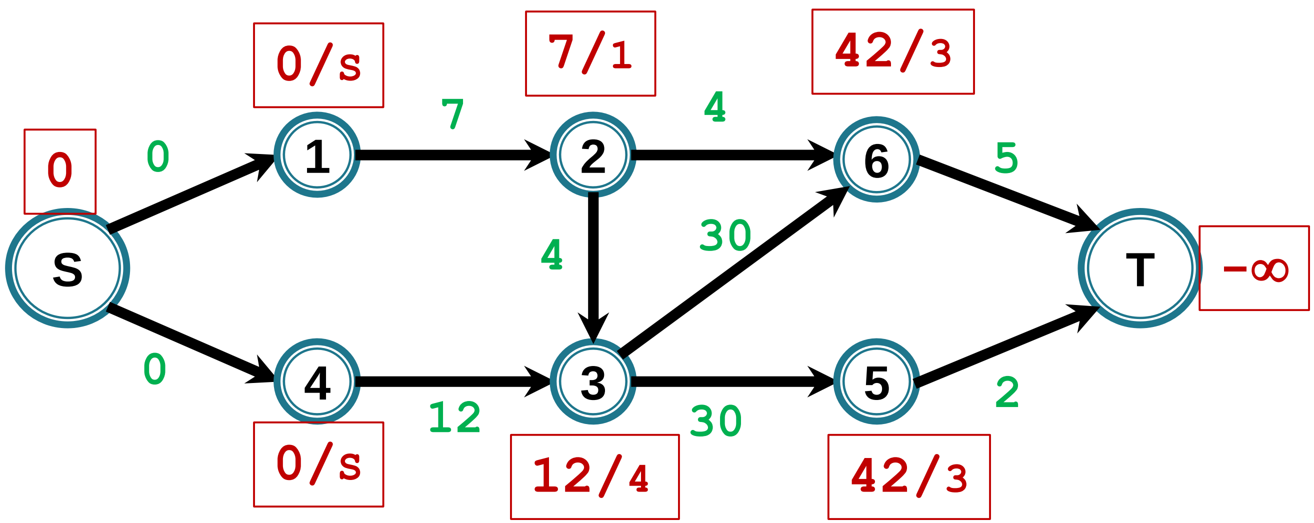
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



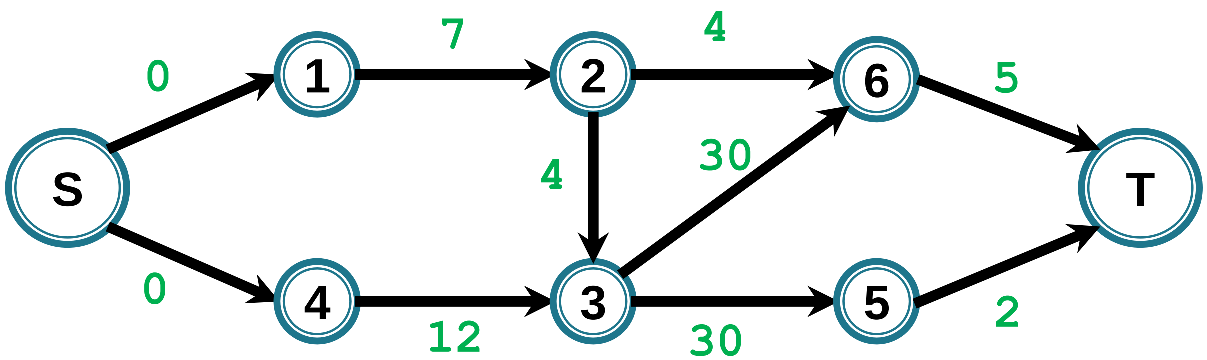
Drumuri critice



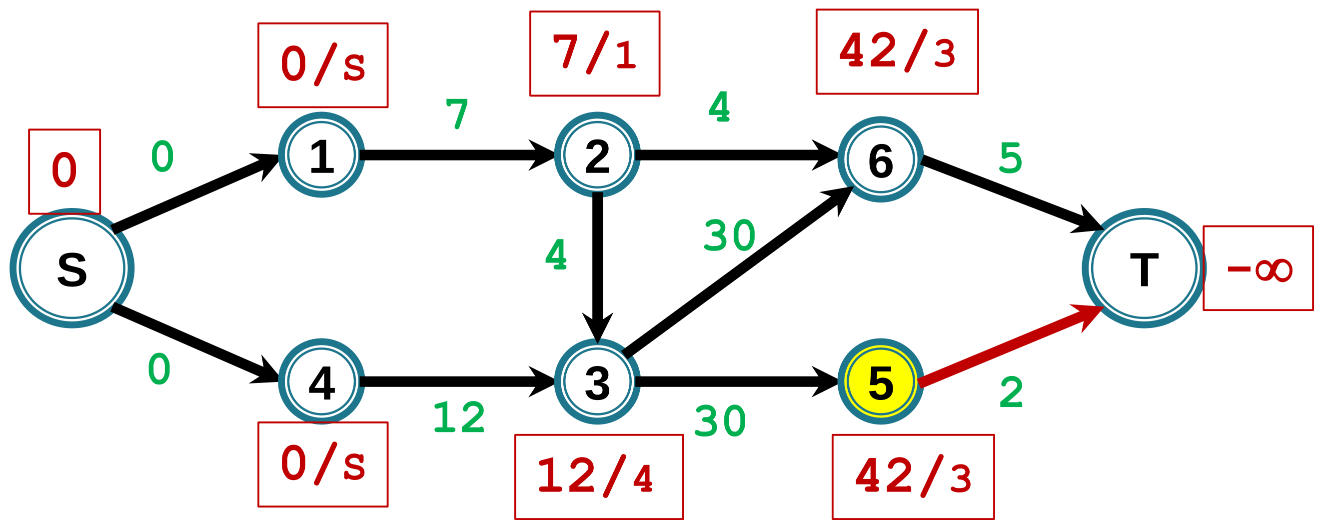
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



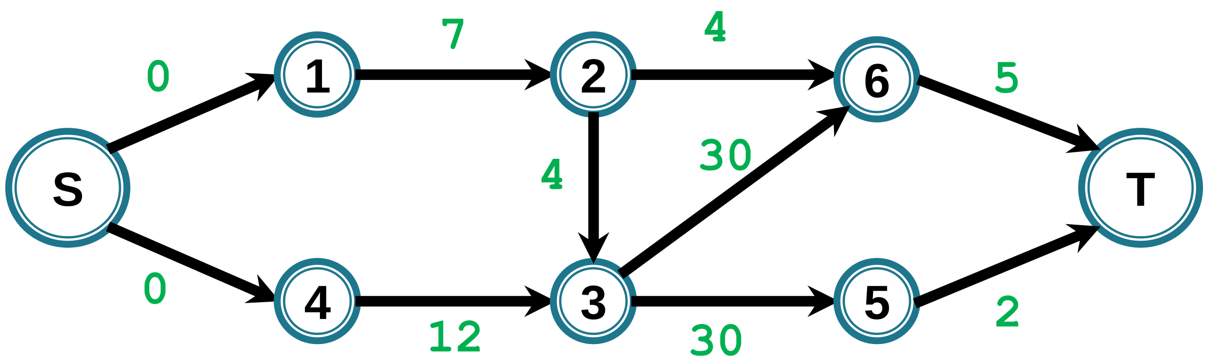
Drumuri critice



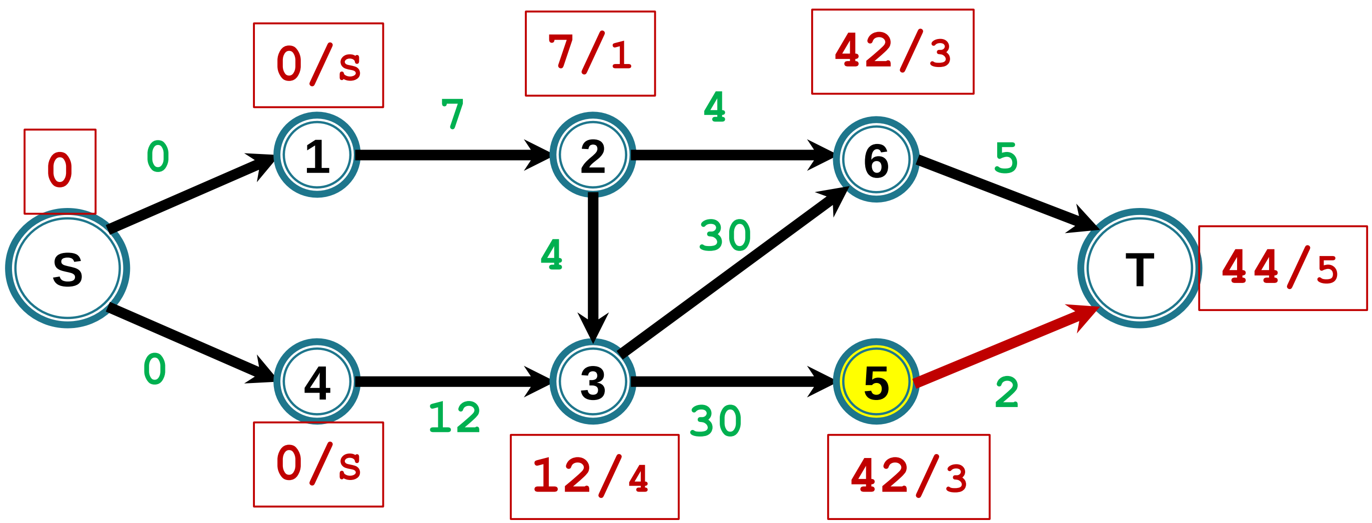
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



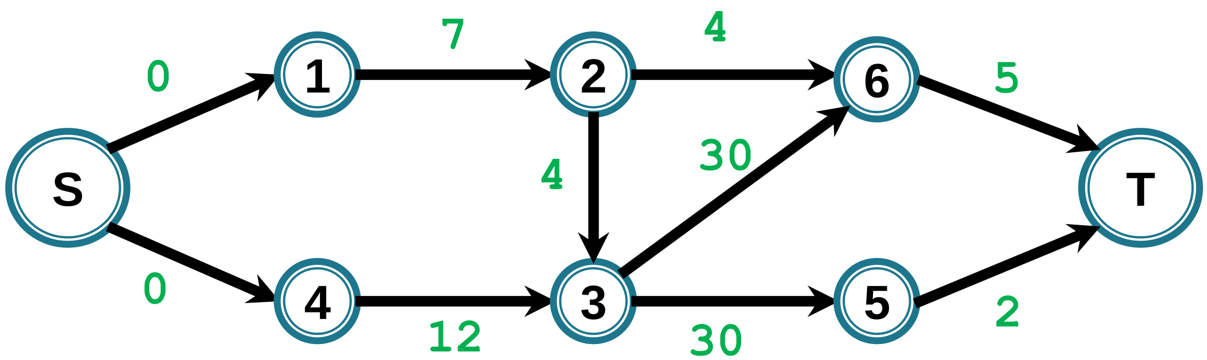
Drumuri critice



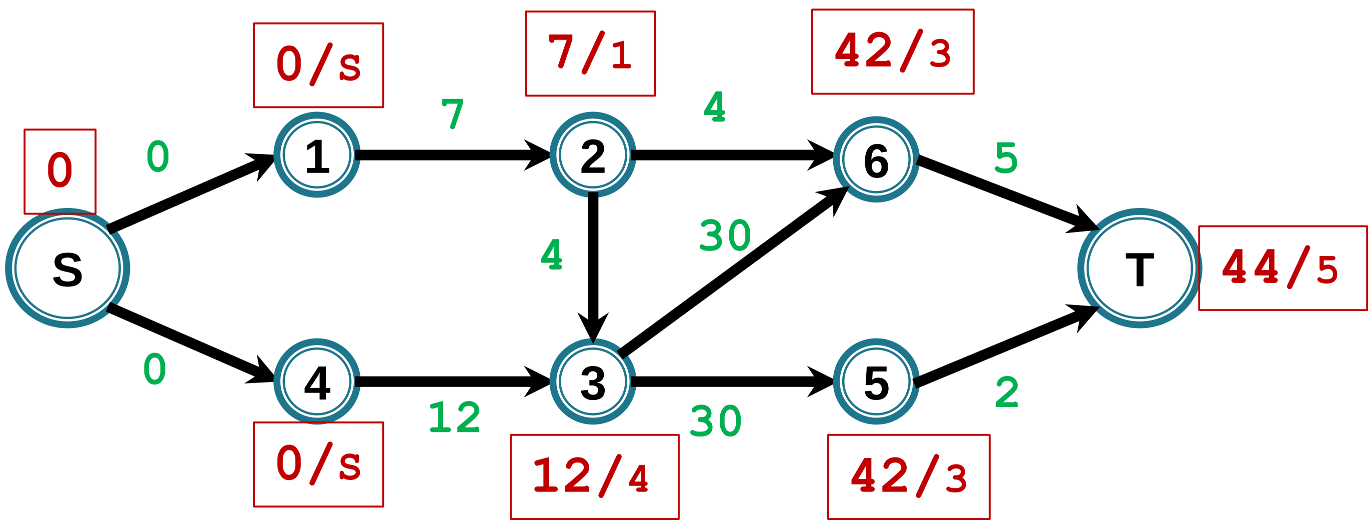
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



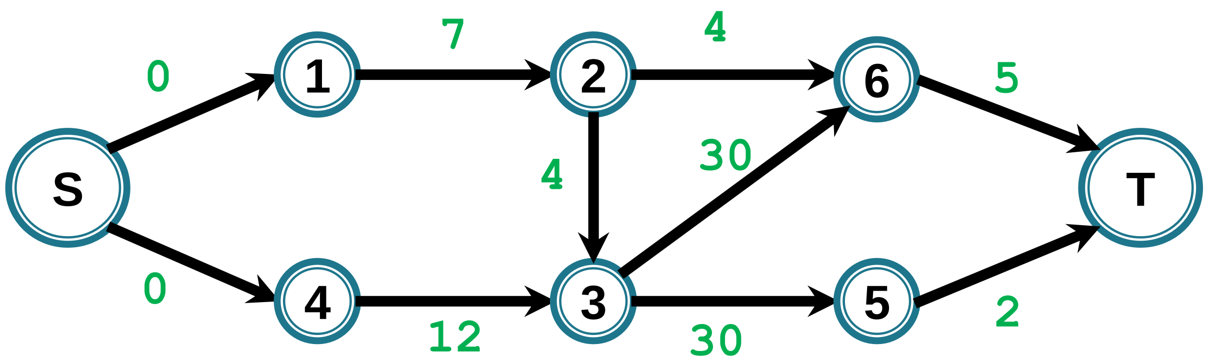
Drumuri critice



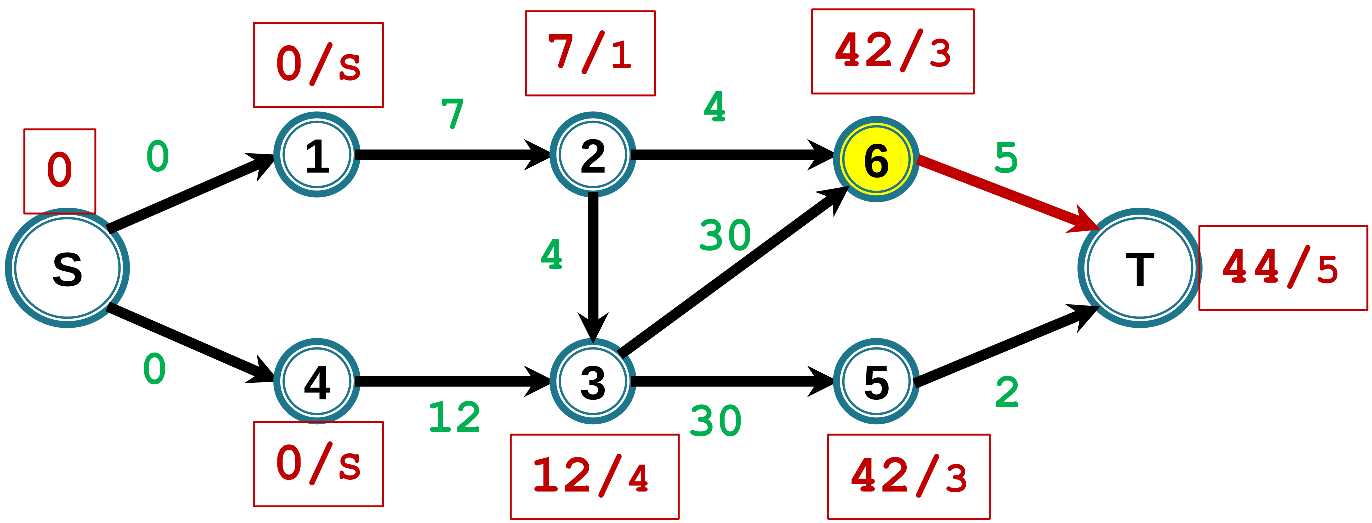
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



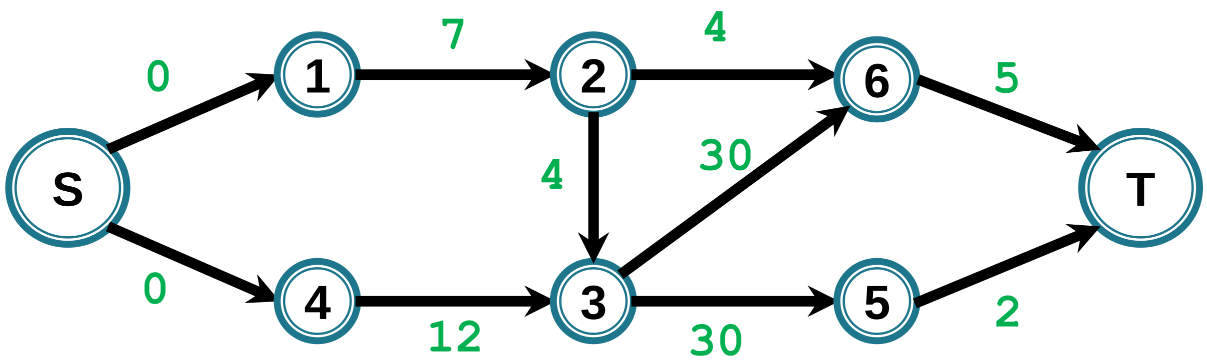
Drumuri critice



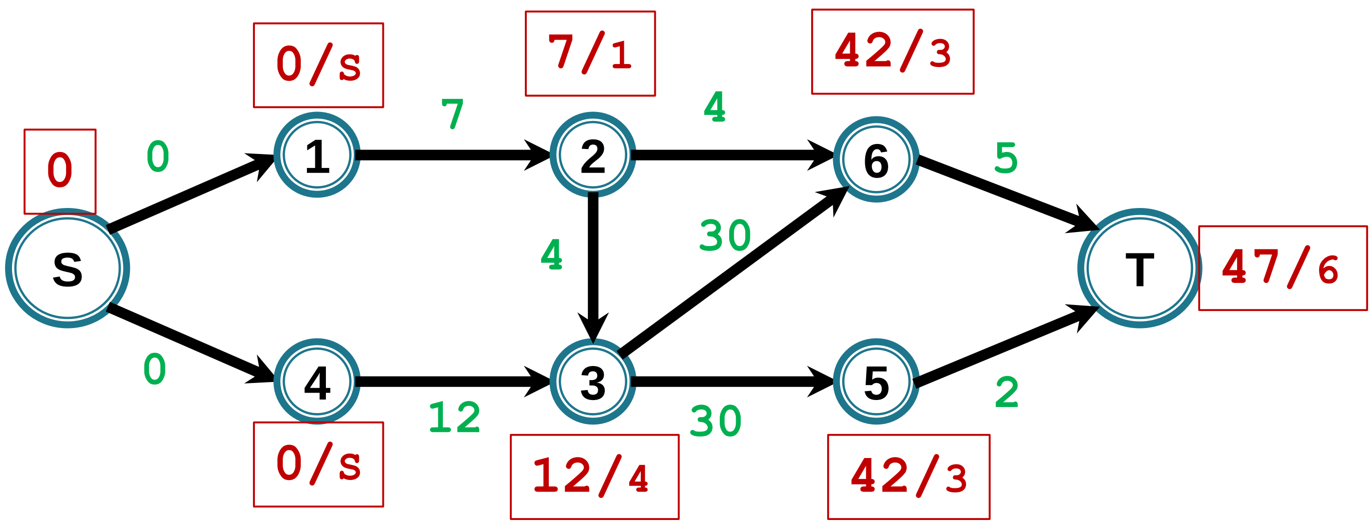
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



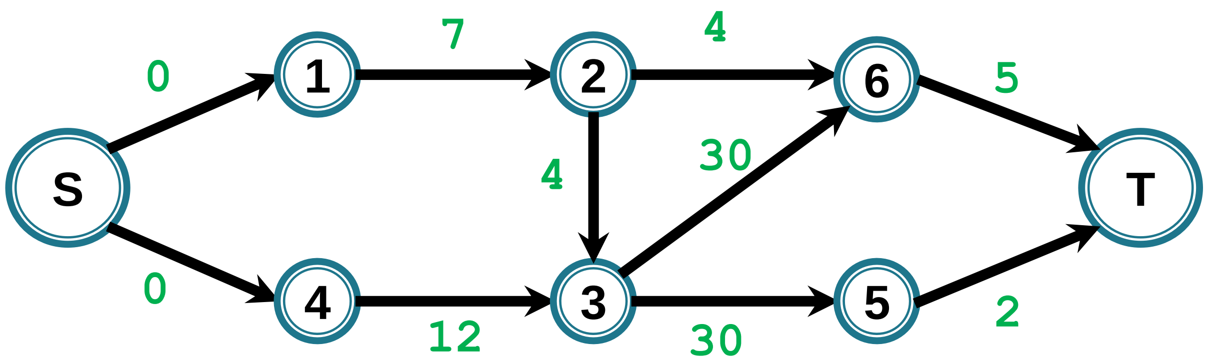
Drumuri critice



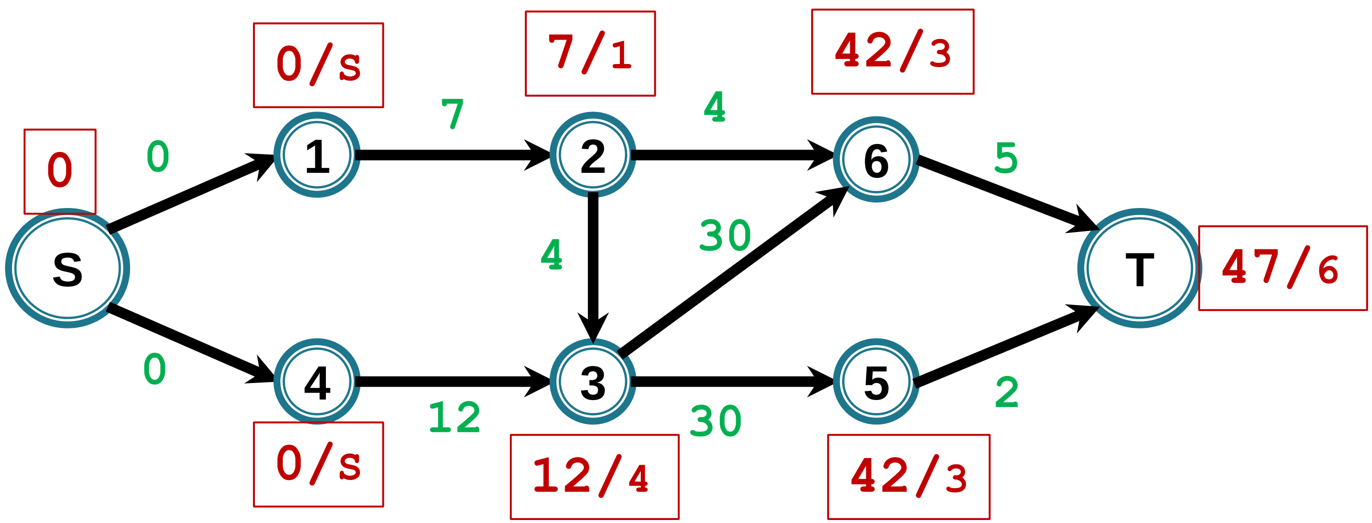
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



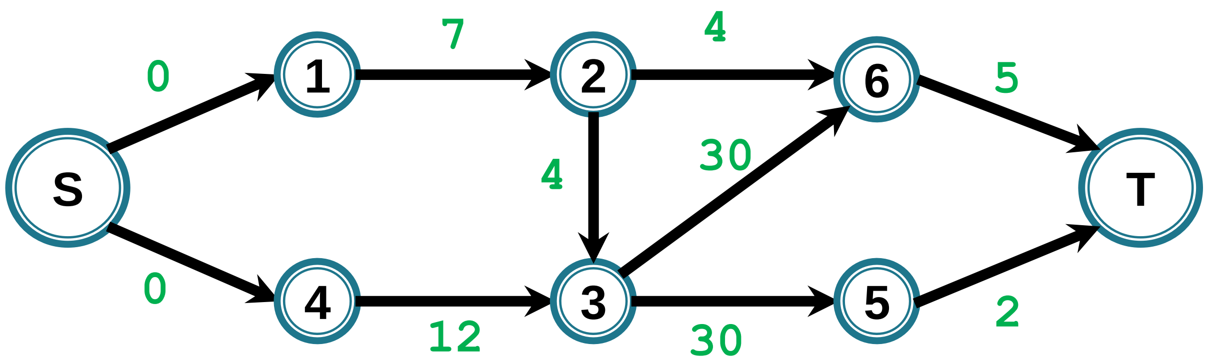
Drumuri critice



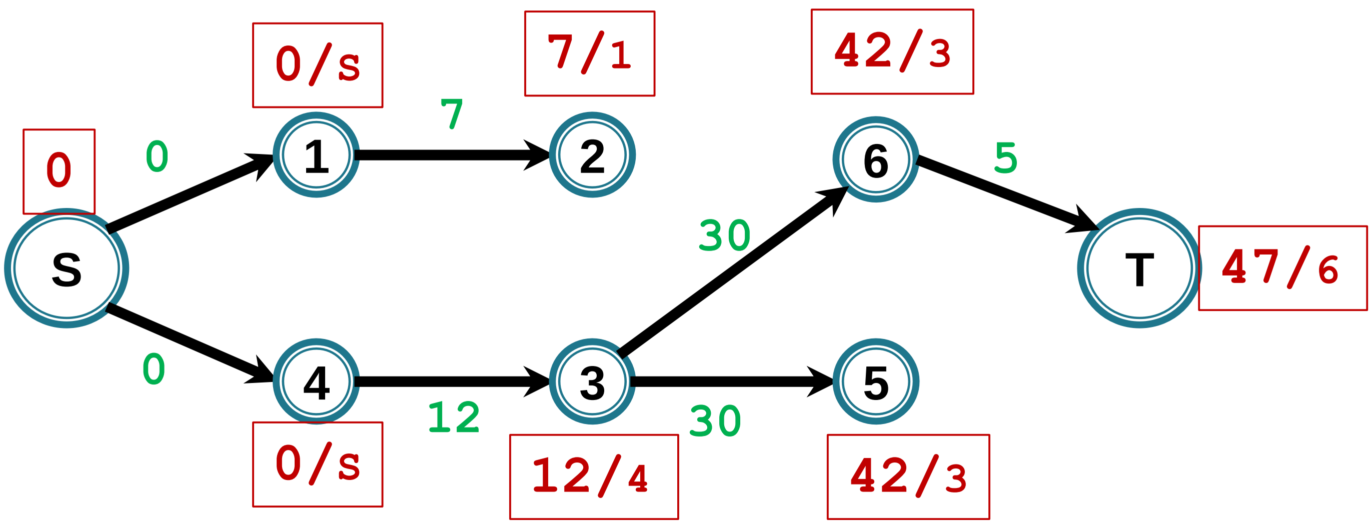
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



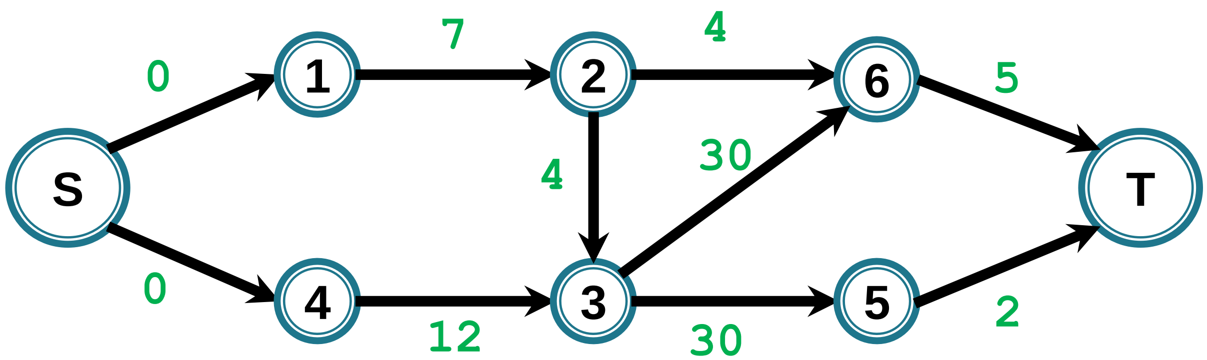
Drumuri critice



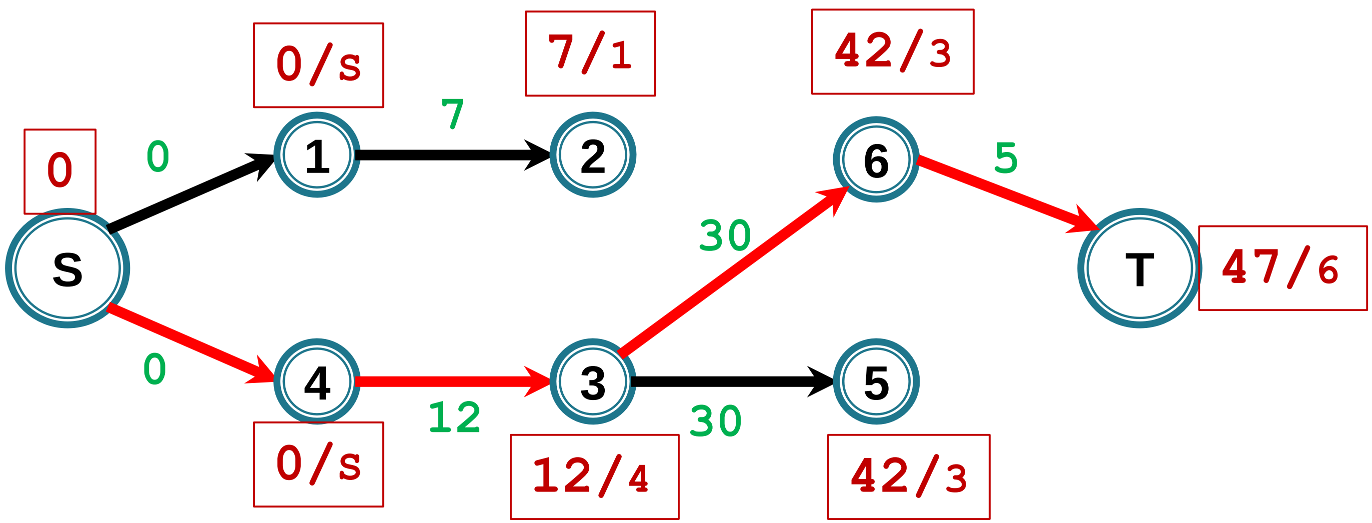
Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



Drumuri critice

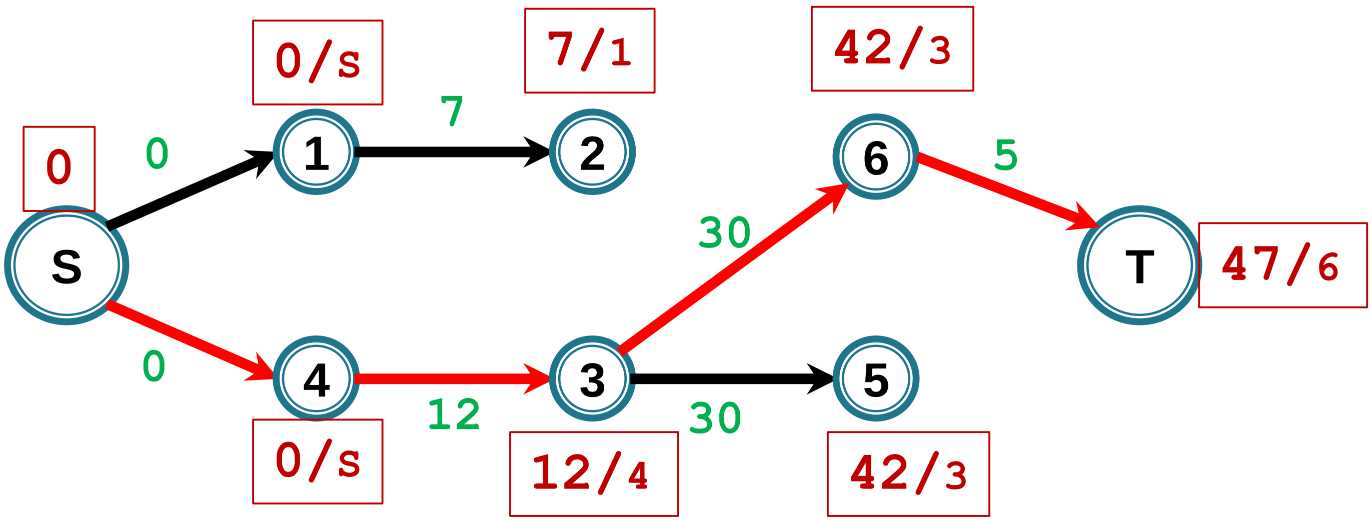


Ordine de calcul distanțe: S, 1, 4, 2, 3, 5, 6, T



Drum critic ⇒ succesiune de activități care determină durata proiectului

Drumuri critice



- **Durata minimă a proiectului: 47**
- Activități critice: 4 3 6
- Intervalele de desfășurare pentru fiecare activitate:

  1: (0, 7)

  2: (7, 8)

  3: (12, 42)

  4: (0, 12)

  5: (12, 42)

  6: (42, 47)

Drumuri maxime

- Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la s la celelalte vârfuri?

Drumuri maxime

Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la S la celelalte vârfuri

- Modificăm astfel doar inițializarea distanțelor (cu -∞ în loc de + ∞) și inversam condiția de la relaxarea arcelor pentru a calcula maxim în loc de minim

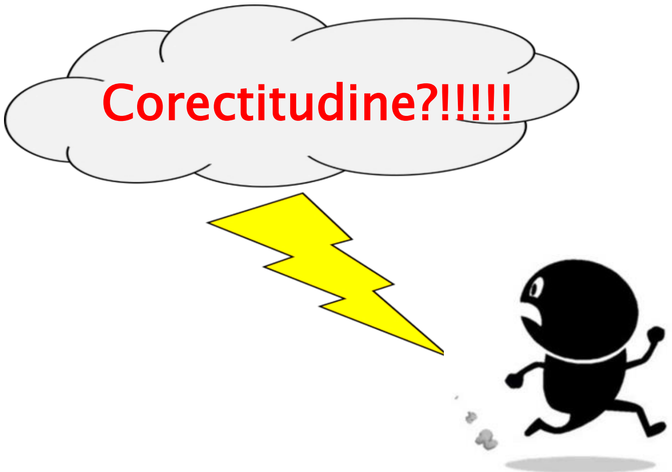
Corectitudine – probabil similar cu Dijkstra?!!

Drumuri maxime

Putem modifica algoritmul lui Dijkstra de determinare de drumuri minime în grafuri (nu neapărat aciclice) a.î. să determine drumuri maxime (elementare) de la S la celelalte vârfuri

- Modificăm astfel doar inițializarea distanțelor (cu -∞ în loc de + ∞) și inversam condiția de la relaxarea arcelor pentru a calcula maxim în loc de minim

- **Corectitudine** – probabil similar cu Dijkstra?!!



Temă – Drumuri de capacitate maximă

- **Problemă: Într-o rețea orientată de comunicație**
  - **w(e) = capacitatea legăturii e** (exp: lățimea de bandă, diametrul unei conducte etc)
  - **Pentru un drum P**

    w(P) = min {w(e) \ e∈ E(P)}

    = cantitatea de informație care se poate transmite de-a lungul drumui P

    = capacitatea minimă a arcelor ce îl compun (pentru ca informația să poată trece prin toate arcele drumului)

Date două vârfuri s și t, să se determine un drum de capacitate maximă de la s la t – Propuneți un algoritm bazat pe o idee similară cu cea din algoritmul lui Dijkstra. Justificați corectitudinea algoritmului propus.