Sortare topologică

Sortare topologică

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

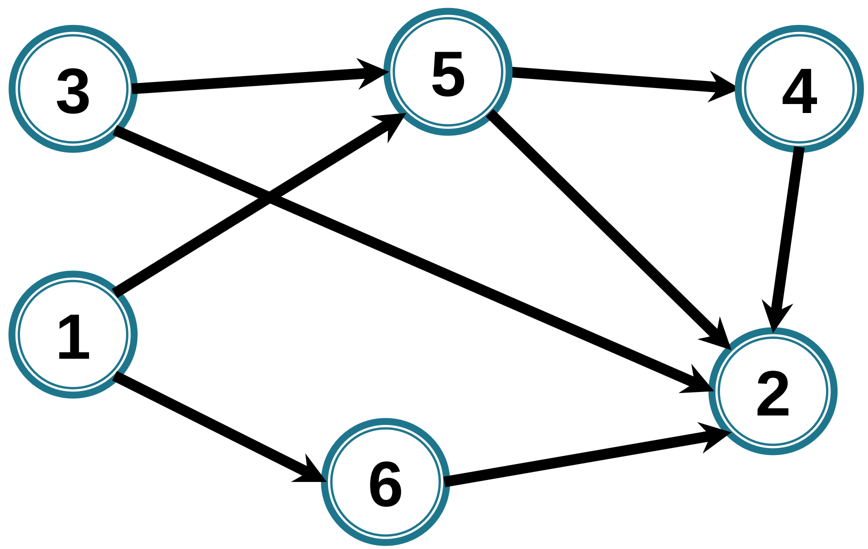
  ordonare a vârfurilor astfel încât dacă $uv \in E$ atunci u se află înaintea lui v în ordonare

  - **Nu este 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

- **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 are o sortare topologică, atunci G este aciclic

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

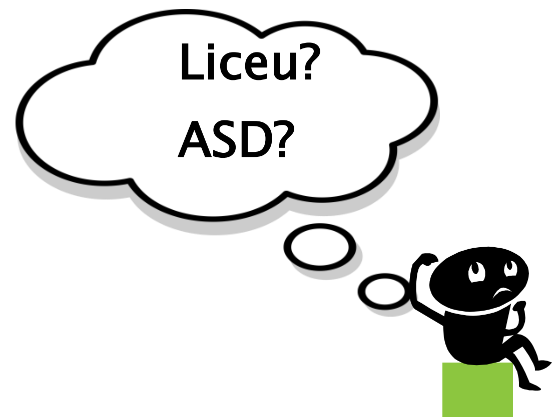
- **Propoziţie.** Dacă G este aciclic atunci G are o sortare topologică

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?**



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 $d^-(v) = 0$

- **Algoritm**

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

Sortare topologică – Algoritm

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

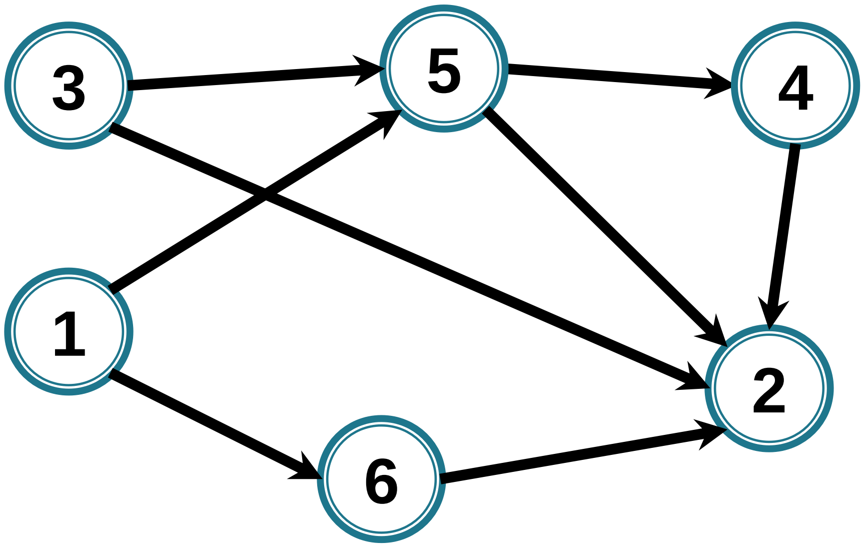
- Algoritm

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

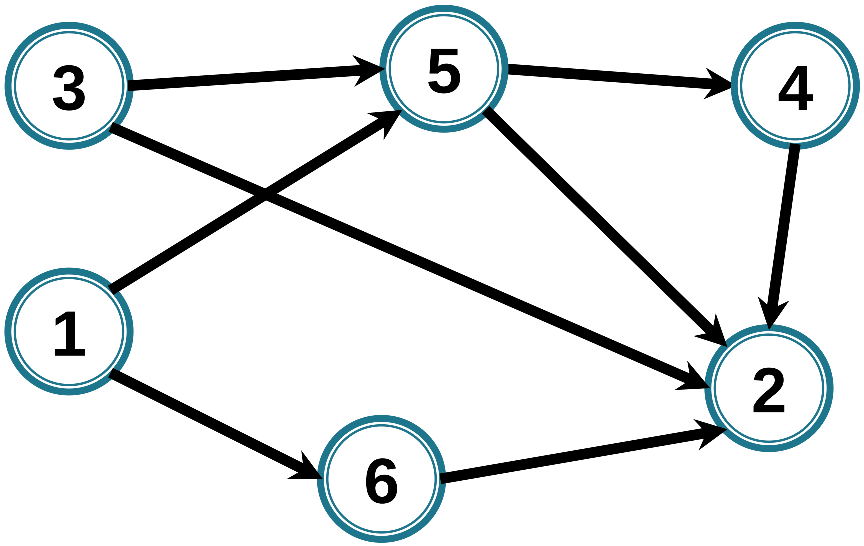
- Implementare – similar BF
  - Pornim cu toate vârfurile cu grad intern 0 și le adăugăm într-o coadă
  - Repetăm:
    - extragem un vârf din coadă
    - îl eliminăm din graf (= scădem gradele interne ale vecinilor, nu îl eliminăm din reprezentare)
    - adăugăm în coadă vecinii al căror grad intern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

Exemplu

Sortare topologică

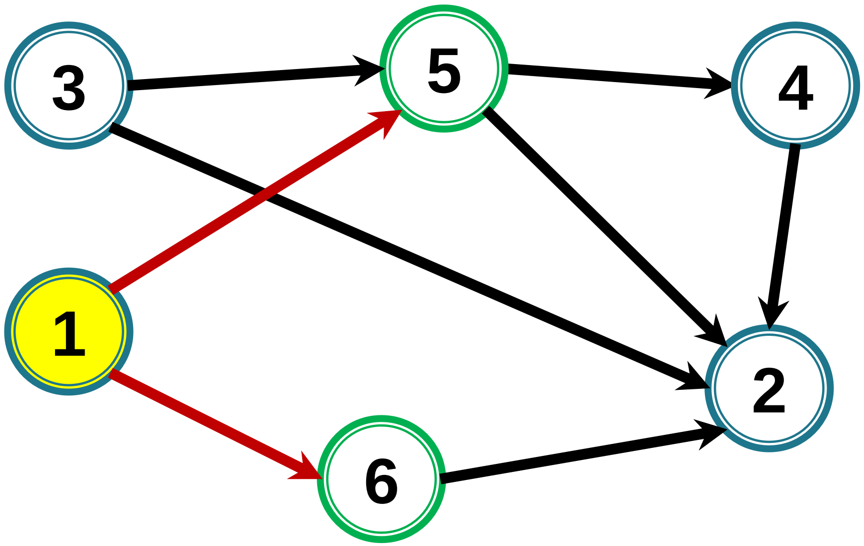


Sortare topologică



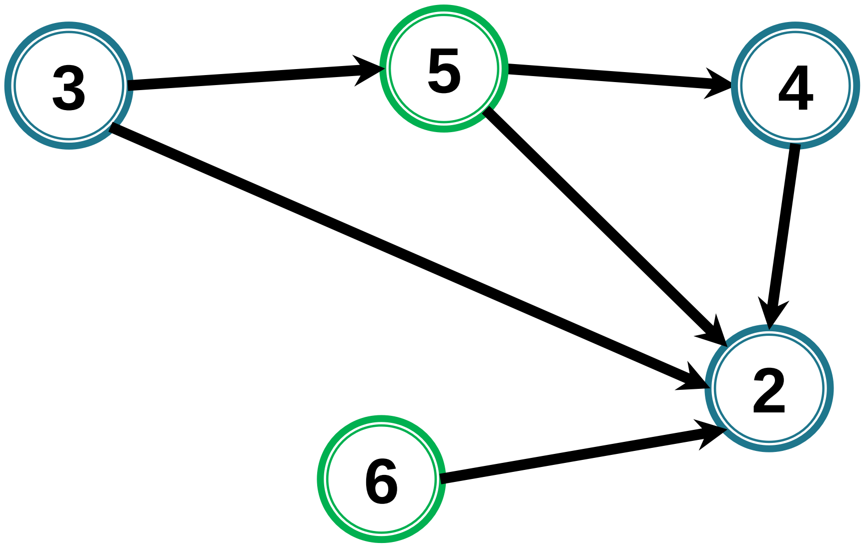
C: 1 3

Sortare topologică



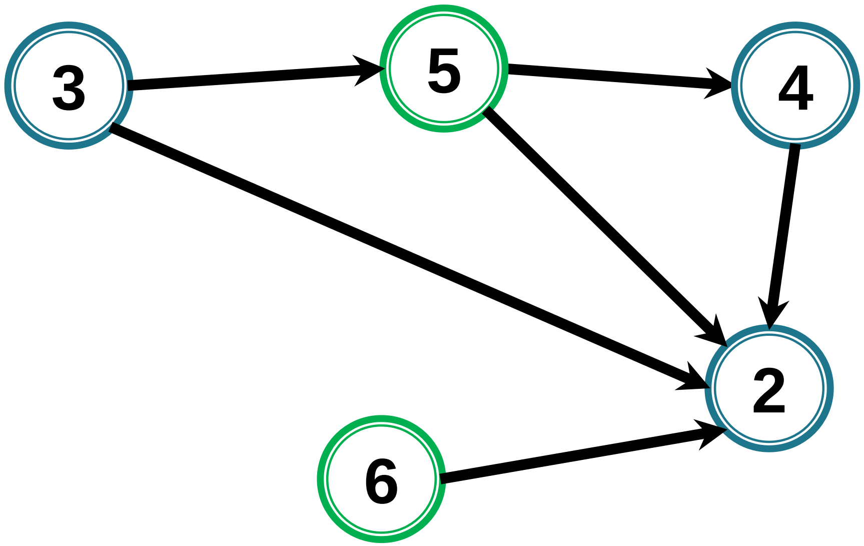
C: 1 3

Sortare topologică



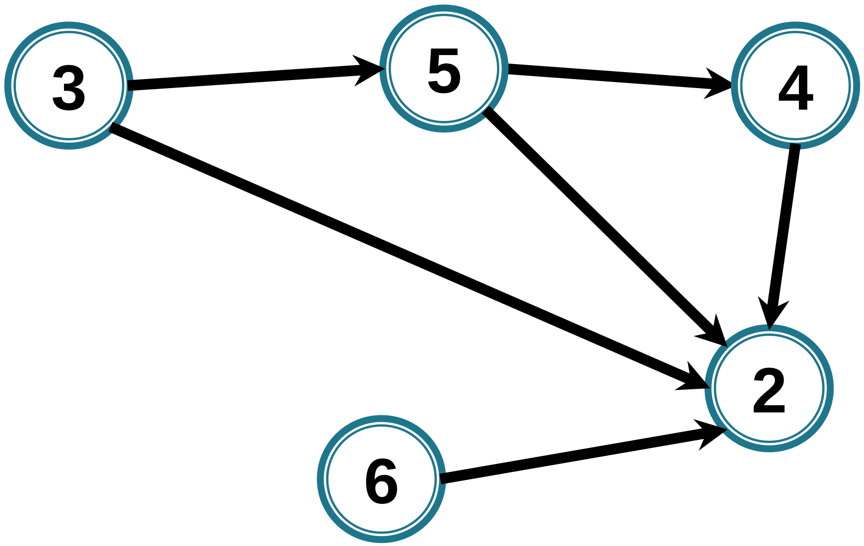
C: 1 3

Sortare topologică



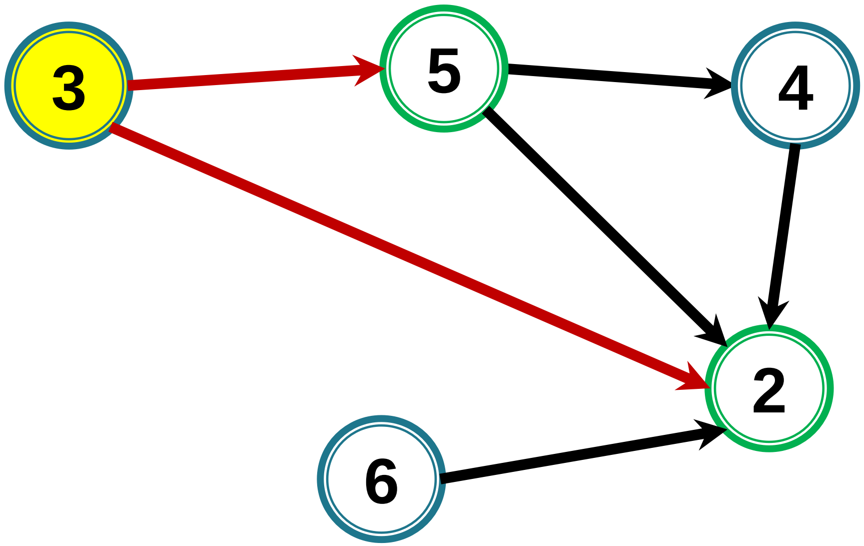
C: 1 3 6

Sortare topologică



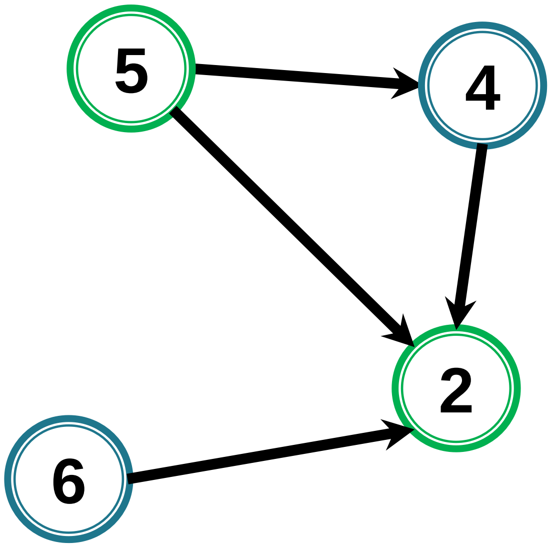
C: 1 3 6

Sortare topologică



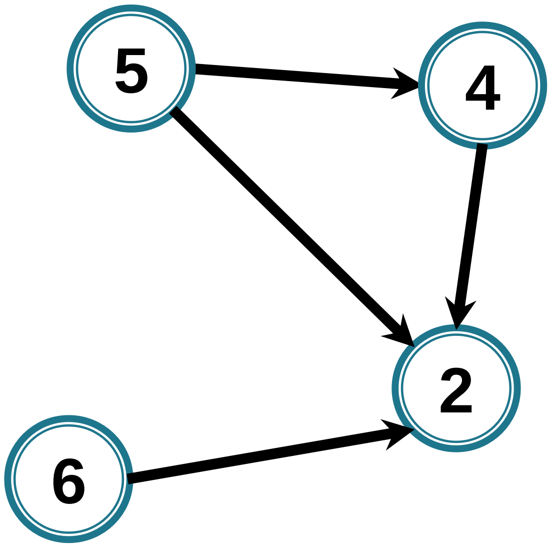
C: 1 3 6

Sortare topologică



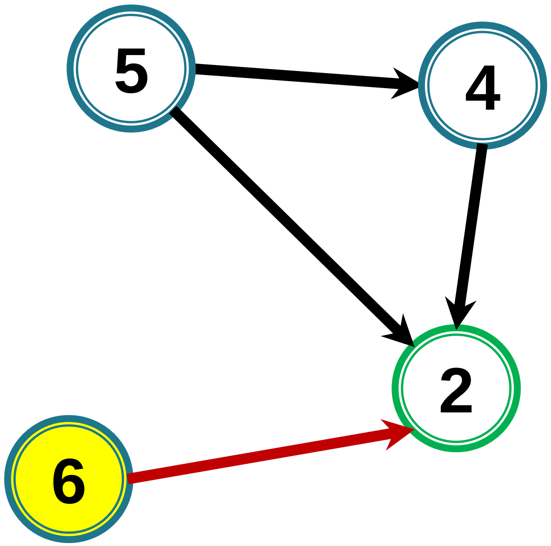
C: 1 3 6

Sortare topologică



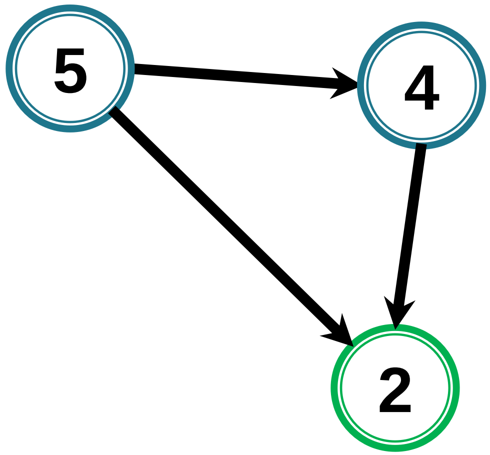
C: 1 3 6 5

Sortare topologică



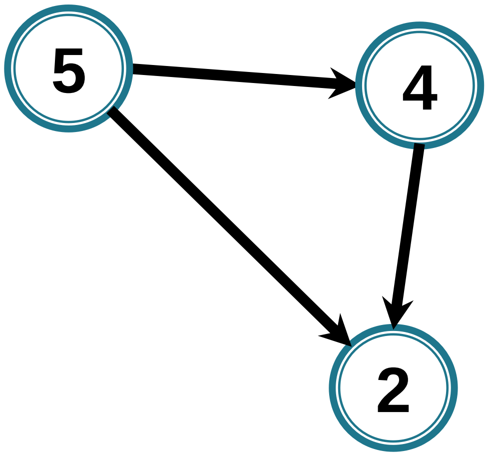
C: 1 3 6 5

Sortare topologică



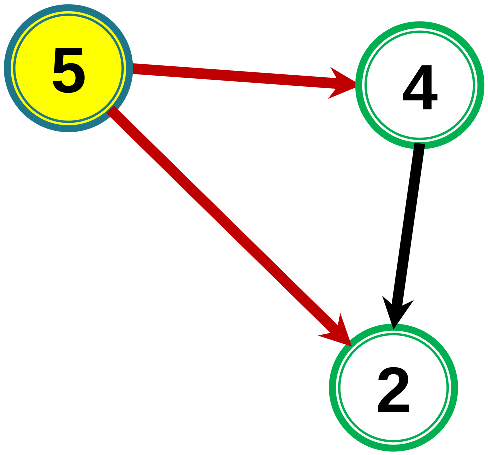
C: 1 3 6 5

Sortare topologică



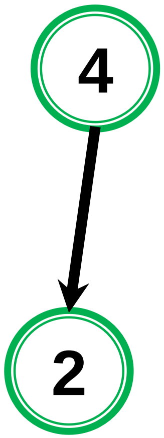
C: 1 3 6 5

Sortare topologică



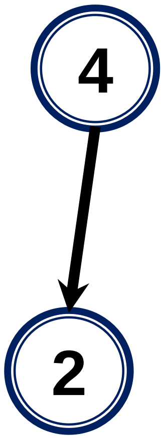
C: 1 3 6 5

Sortare topologică



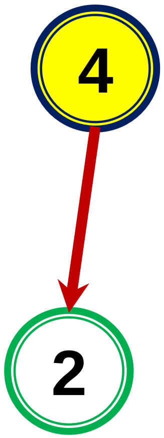
C: 1 3 6 5

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

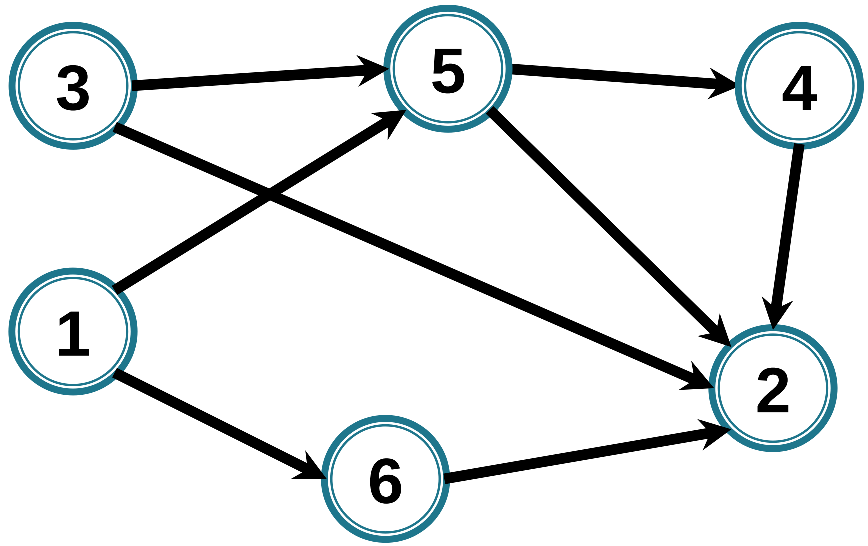
Sortare topologică

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

- **Suplimentar** – există un algoritm bazat pe DF, pornind de la următoarea **observație:**
  - Dacă **final**[u] = momentul la care a fost <u>**finalizat**</u> vârful u în parcurgerea DF avem:

    $$\mathbf{uv \in E \;\Rightarrow\; final[u] > final[v]}$$

  -

Sortare topologică – Alt algoritm

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

    $$uv \in E \Rightarrow \texttt{final[u]} > \texttt{final[v]}$$

  - Atunci sortare topologică = sortare descrescătoare în raport cu final

Sortare topologică - Alt algoritm

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

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

    $$\mathbf{uv} \in \mathbf{E} \quad \Rightarrow \quad \mathbf{final[u]} > \mathbf{final[v]}$$

  - Atunci sortare topologică = sortare descrescătoare în raport cu final

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

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