Drumuri minime între toate perechile de vârfuri

Algoritmul Floyd-Warshall

Problema drumurilor minime <u>între toate perechile de vârfuri</u>

Se dă:

- un graf orientat ponderat G= (V, E, w)

Pentru oricare două vârfuri x și y al lui G să se determine distanța de la x la y și un drum minim de la x la y

Ponderile pot fi și <u>negative</u> dar NU există circuite cu cost negativ în G

Soluția 1

Se aplică algoritmul lui Dijkstra/Bellman –Ford pentru fiecare vârf x

- **Soluția 2**

Algoritmul Floyd-Warshall

Floyd-Warshall

- Fie W= $(w_{ij})_{i,j=1..n}$ **matricea costurilor** grafului G:

$$w_{ij} = \begin{cases} 0, \text{ daca } i = j \\ w(i,j), \text{ daca } ij \in E \\ \infty, \text{ daca } ij \notin E \end{cases}$$

- Vrem să calculăm **matricea distanțelor** D = $(d_{ij})_{i,j=1..n}$:

$$\mathbf{d_{ij} = \delta(i, j)}$$

Floyd–Warshall

- Fie W= $(w_{ij})_{i,j=1..n}$ matricea costurilor grafului G:

$$w_{ij} = \begin{cases} 0, \text{ daca } i = j \\ w(i,j), \text{ daca } ij \in E \\ \infty, \text{ daca } ij \notin E \end{cases}$$

- Vrem să calculăm matricea distanțelor D = $(d_{ij})_{i,j=1..n}$:

$$\mathbf{d_{ij} = \delta(i, j)}$$

- **Observație:** w_{ij} = costul minim al unui i–j drum fără vârfuri intermediare (cu cel mult un arc)

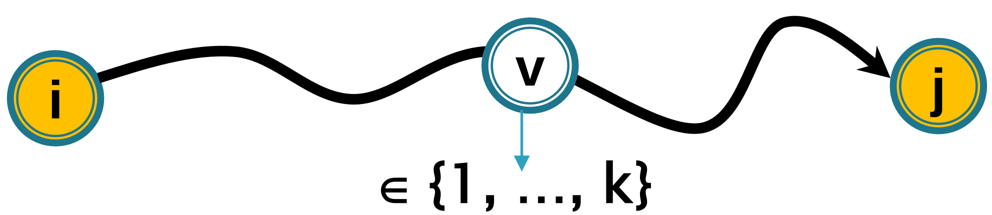
Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall:**

Programare Dinamică – Subprobleme:

Pentru k = 1, 2, …, n calculăm pentru oricare două vârfuri i, j:

costul minim al unui drum (elementar) de la i la j care are ca <u>vârfuri intermediare doar vârfuri din mulțimea</u> {1, 2, …, k}



Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall**:

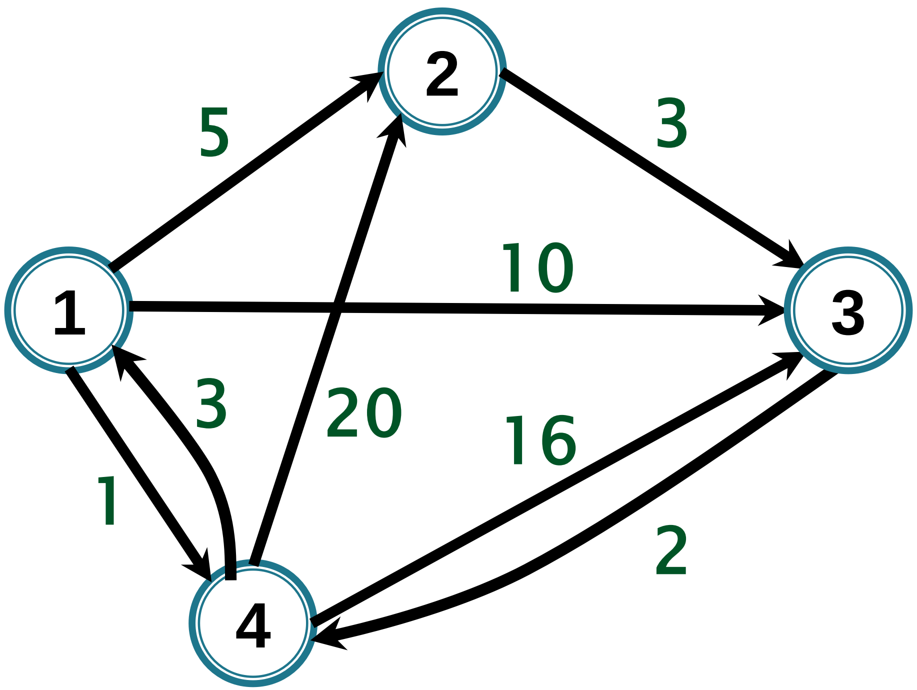
Astfel, pentru k = 1, 2, …, n calculăm **matricea**

$D^{(k)} = (d^k_{ij})_{i,j=1..n}$:

d^k_{ij} = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,…, k}

Floyd-Warshall

d^k_{ij} = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,..., k}



$d^0_{4,3} =$

$d^1_{4,3} =$

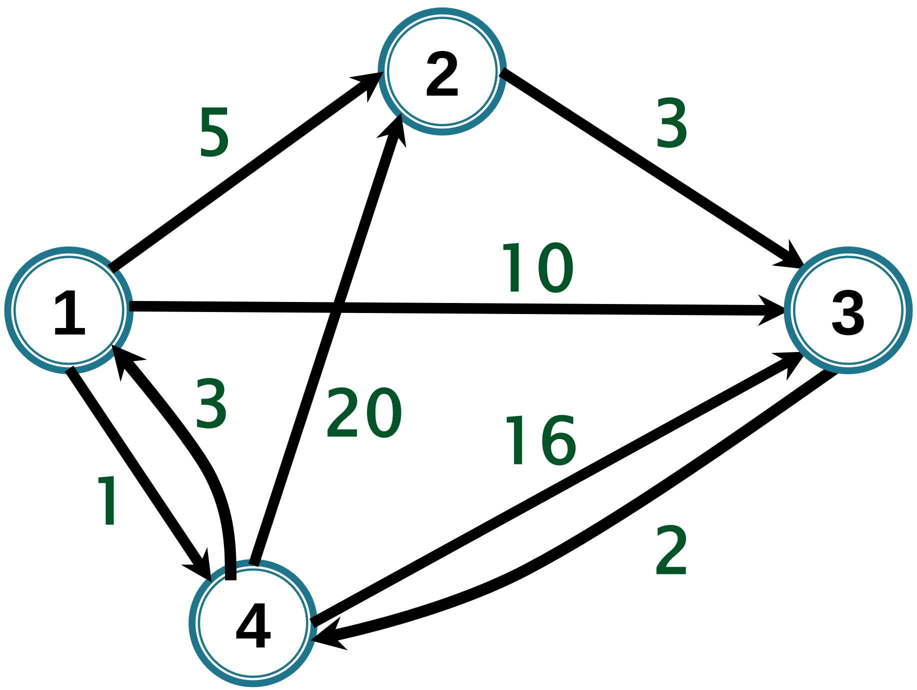
$d^2_{4,3} =$

$d^2_{1,3} =$

$d^4_{1,3} =$

Floyd-Warshall

d^k_{ij} = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,…, k}



$d^0_{4,3} = 16$ arc direct

$d^1_{4,3} = 13$ – drum [4,1,3]

$d^2_{4,3} = 11$ – drum [4,1,2,3]

$d^2_{1,3} = 8$ – drum [1,2, 3]

$d^4_{1,3} = 8$ – drum [1,2, 3]

Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall**:

Astfel, pentru k = 1, 2, …, n calculăm **matricea**

$D^{(k)} = (d^k_{ij})_{i,j=1..n}$:

$\mathbf{d^k_{ij}}$ = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,…, k}

- **Iniţializare:** 

Floyd–Warshall

- **Ideea algoritmului Floyd–Warshall**:

Astfel, pentru k = 1, 2, …, n calculăm **matricea**

$D^{(k)} = (d^k_{ij})_{i,j=1..n}$:

$\mathbf{d^k_{ij}}$ = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,…, k}

- **Iniţializare:** $\mathbf{D^{(0)} = W}$

Care este matricea distanțelor?

Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall**:

Astfel, pentru k = 1, 2, …, n calculăm **matricea**

$D^{(k)} = (d^k_{ij})_{i,j=1..n}$:

$\mathbf{d^k_{ij}}$ = costul minim al unui drum de la i la j care are vârfurile intermediare în {1, 2,…, k}

- **Iniţializare:** $\mathbf{D^{(0)} = W}$

- **Avem** $\mathbf{D^{(n)} = D}$

Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall**:

Pentru a **reține și un drum minim** 

Floyd-Warshall

- **Ideea algoritmului Floyd-Warshall**:

Pentru a **reține și un drum minim**

- **matrice de predecesori** $P^{(k)} = (p^k_{ij})_{i,j=1..n}$:

  - linia i corespunde vârfului de start i

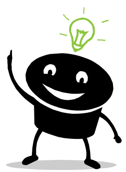
p^k_{ij} = predecesorul lui j pe drumul minim curent găsit de la i la j care are vârfurile intermediare în {1, 2,..., k}

Floyd-Warshall

- **Cum calculăm elementele matricei** $D^{(k)}$?

Floyd-Warshall

- **Cum calculăm elementele matricei** $D^{(k)}$?

 **Relaţie de recurenţă**

Floyd-Warshall - relația de recurență

Fie P un drum (elementar) de cost minim de la i la j cu vârfurile intermediare în mulțimea {1,2,..., k}

- **Dacă k nu este vârf intermediar în P** =>

P un drum (elementar) de cost minim de la i la j cu vârfurile intermediare în mulțimea {1,2,..., k-1} :

$$\Rightarrow w(P) = \mathbf{d}^{k-1}_{ij}$$

Floyd–Warshall – relația de recurență

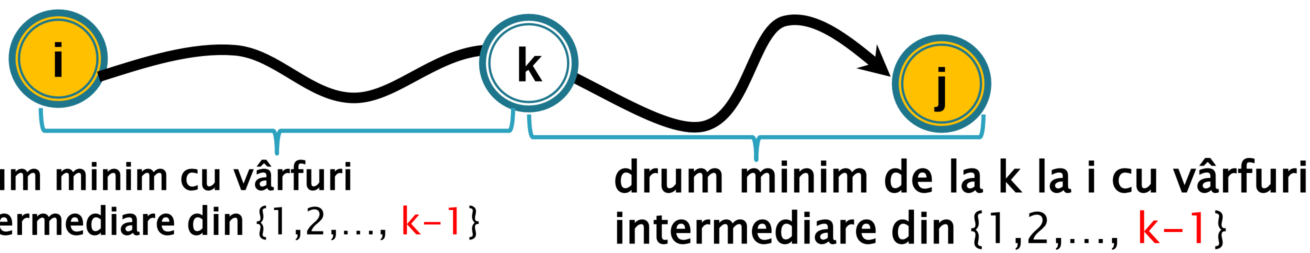
Fie P un drum (elementar) de cost minim de la i la j cu vârfurile intermediare în mulțimea {1,2,..., k}

- **Dacă k nu este vârf intermediar în P** =>

  P un drum (elementar) de cost minim de la i la j cu vârfurile intermediare în mulțimea {1,2,..., k−1} :

  $$=> w(P) = d^{k-1}_{ij}$$

- **Dacă vârful k este vârf intermediar al lui P**



$$=> w(P) = d^{k-1}_{ik} + d^{k-1}_{kj}$$

Floyd–Warshall – relația de recurență

Fie P un drum (elementar) de cost minim de la i la j cu vârfurile intermediare în mulțimea {1,2,…, k}

=> $w(P) = d^{k}_{ij} = \min\{d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj}\}$

Floyd-Warshall

- Se obține astfel relația

$$d^{k}_{ij} = \min\{d^{k-1}_{ij},\ d^{k-1}_{ik} + d^{k-1}_{kj}\}$$

- Observații
  - Avem

$$d^{k}_{ik} = d^{k-1}_{ik}$$

$$d^{k}_{kj} = d^{k-1}_{kj}$$

de aceea în implementarea algoritmului putem folosi o singură matrice

Floyd-Warshall

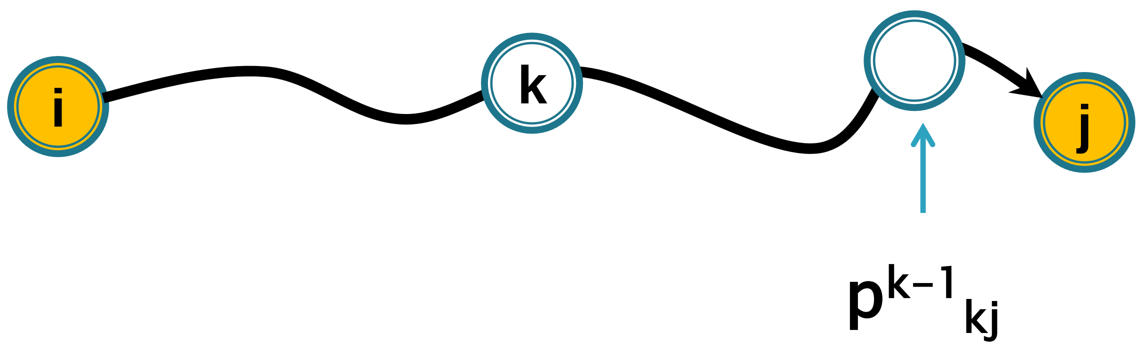
- Se obține astfel relația

$$d^k_{ij} = \min \{d^{k-1}_{ij}, d^{k-1}_{ik} + d^{k-1}_{kj}\}$$

- Observații

Când de actualizează $d^k_{ij} = d^{k-1}_{ik} + d^{k-1}_{kj}$ trebuie actualizat și p^k_{ij}

$$p^k_{ij} = p^{k-1}_{kj}$$



Implementare

Floyd-Warshall

- Conform observaţiilor anterioare, putem folosi o unică matrice D

- **Iniţializare**

$$d[i][j] = w(i,j) \text{ – costul arcului } (i,j)$$

$$p[i][j] = \begin{cases} i, \text{ daca } ij \in E \\ 0, \text{ altfel} \end{cases}$$

Floyd(G, w)

```
for(i=1;i<=n;i++)  //initial d = matricea costurilor
    for(j=1;j<=n;j++){
        d[i][j]=w[i][j];
        if(w[i][j]== ∞)
             p[i][j]=0;
        else
            p[i][j]=i;
    }
```

Floyd(G, w)

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++){
        d[i][j]=w[i][j];
        if(w[i][j]== ∞)
            p[i][j]=0;
        else
            p[i][j]=i;
    }
for(k=1;k<=n;k++)//varfuri intermediare
```

Floyd(G, w)

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++){
        d[i][j]=w[i][j];
        if(w[i][j]== ∞)
            p[i][j]=0;
        else
            p[i][j]=i;
    }
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
```

Floyd(G, w)

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++){
        d[i][j]=w[i][j];
        if(w[i][j]== ∞)
            p[i][j]=0;
        else
            p[i][j]=i;
    }
for(k=1;k<=n;k++)
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(d[i][j]>d[i][k]+d[k][j]){
                d[i][j]=d[i][k]+d[k][j];
                p[i][j]=p[k][j];
            }
```

Floyd-Warshall

- **Ieșire:** matricea d = **matricea distanțelor minime**

- Afișarea unui drum de la i la j, **daca d[i][j]<∞**, se face folosind matricea p

Floyd–Warshall

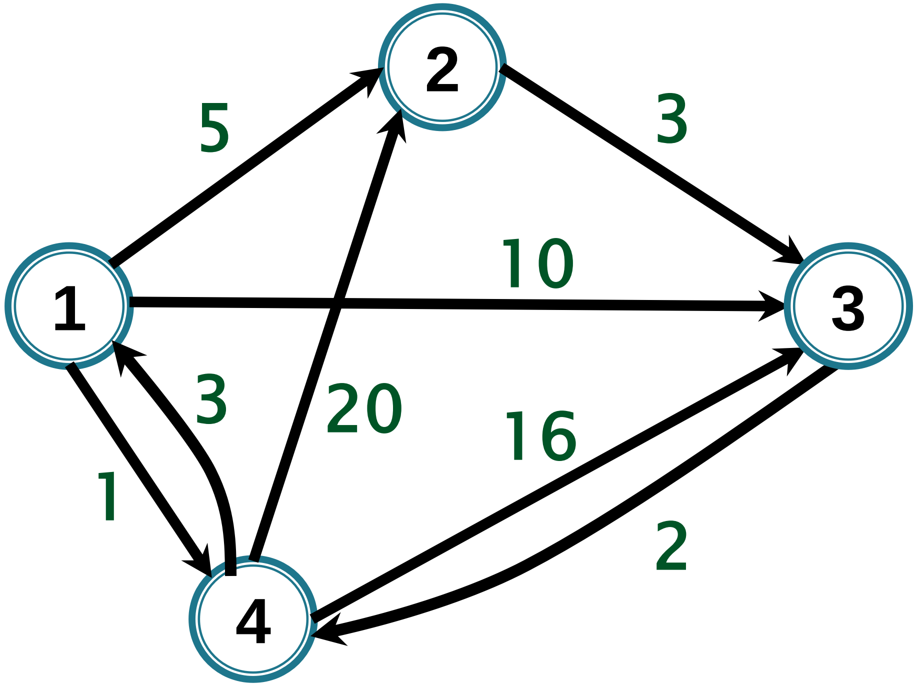
- **Ieșire:** matricea d = **matricea distanțelor**

- Afișarea unui drum de la i la j, **daca d[i][j]<∞**, se face folosind matricea p

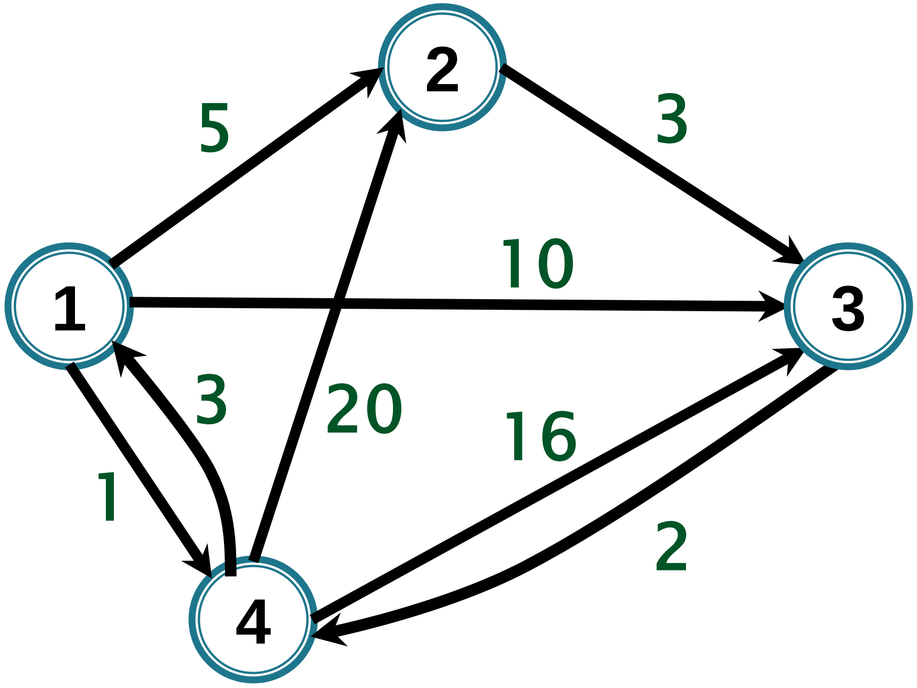
```
void drum(int i,int j){
      if(i!=j)
            drum(i,p[i][j]);
      cout<<j<<" ";
 }
```

Floyd-Warshall

Complexitate – $O(n^3)$



2
5
3
10
1
3
3
20
16
1
2
4

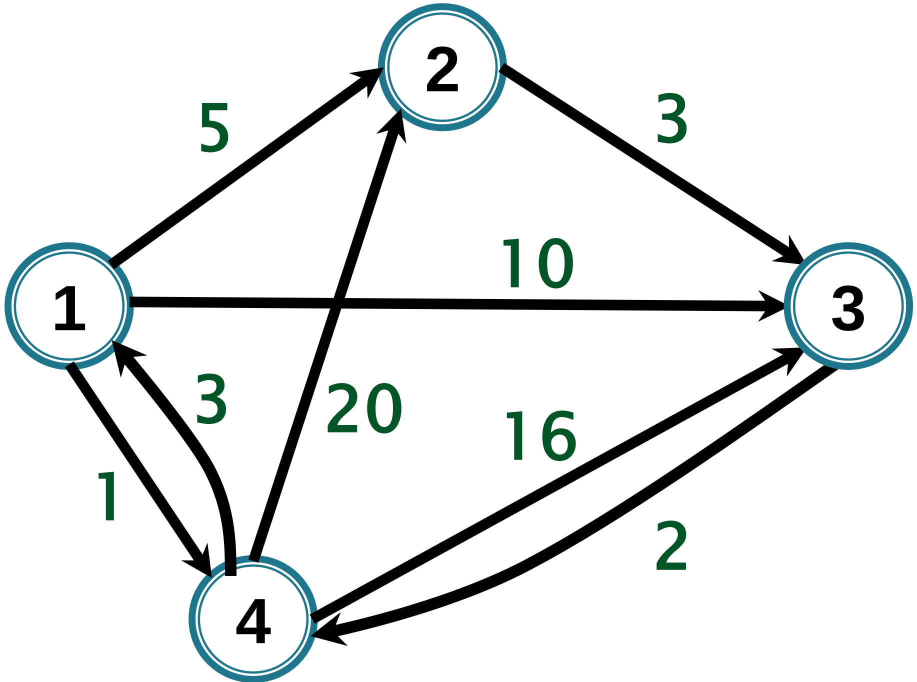


W=d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	20	16	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	4	4	0



W=d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	20	16	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	4	4	0

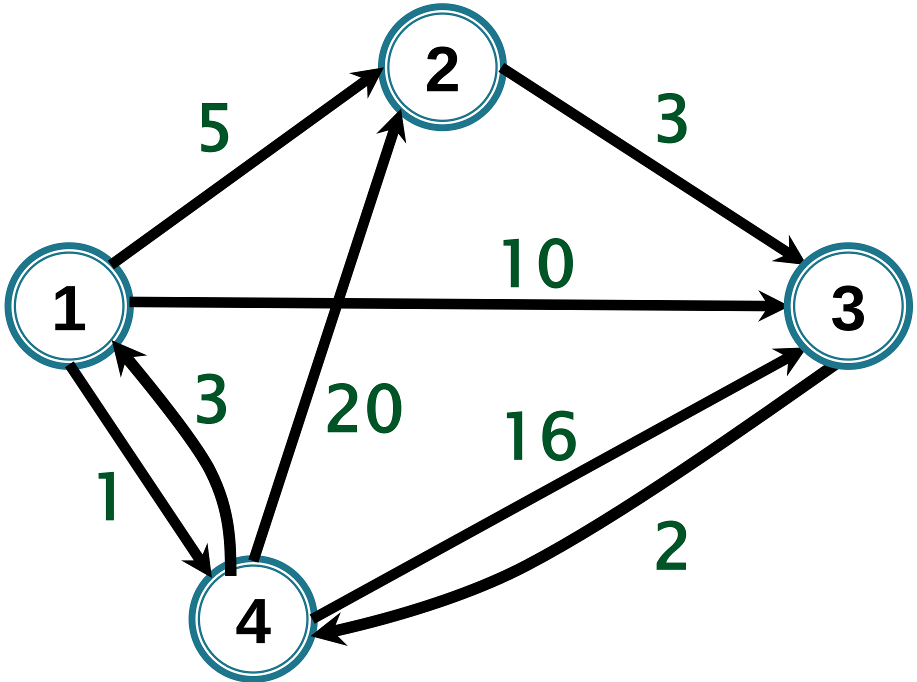
k=1

d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	8	13	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	1	1	0



W=d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	20	16	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	4	4	0

k=1

d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	8	13	0

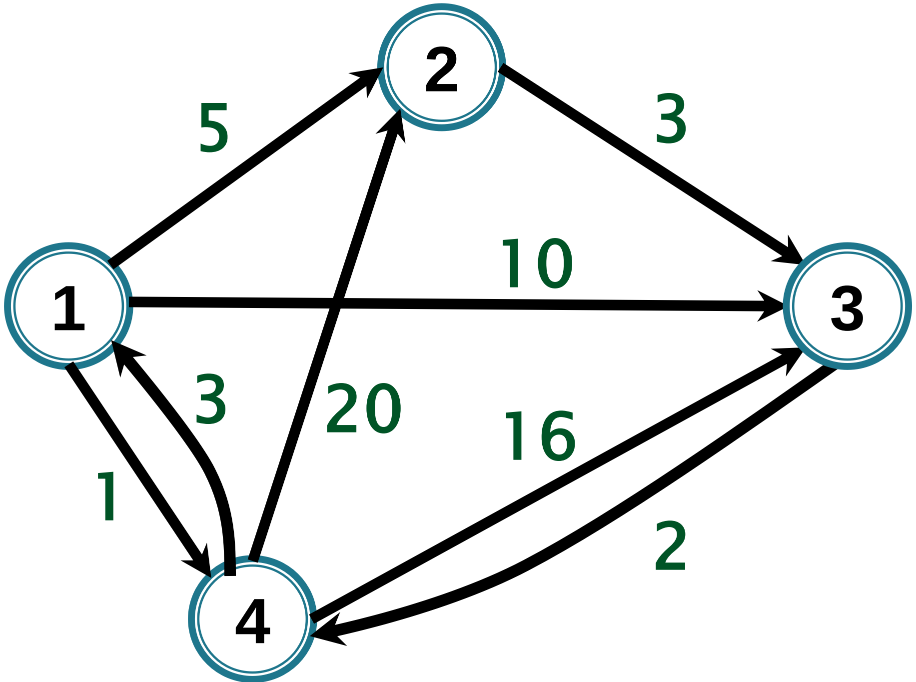
p=

0	1	1	1
0	0	2	0
0	0	0	3
4	1	1	0

k=2

0	5	8	1
∞	0	3	∞
∞	∞	0	2
3	8	11	0

0	1	2	1
0	0	2	0
0	0	0	3
4	1	2	0



W=d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	20	16	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	4	4	0

k=1

d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	8	13	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	1	1	0

k=2

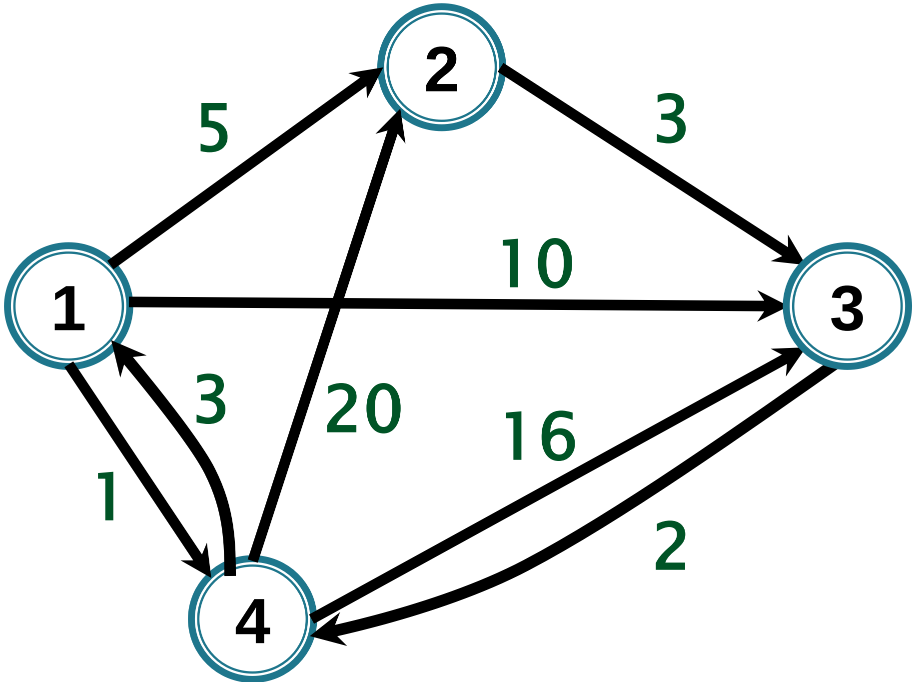
0	5	8	1
∞	0	3	∞
∞	∞	0	2
3	8	11	0

0	1	2	1
0	0	2	0
0	0	0	3
4	1	2	0

k=3

0	5	8	1
∞	0	3	5
∞	∞	0	2
3	8	11	0

0	1	2	1
0	0	2	3
0	0	0	3
4	1	2	0



W=d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	20	16	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	4	4	0

k=1

d=

0	5	10	1
∞	0	3	∞
∞	∞	0	2
3	8	13	0

p=

0	1	1	1
0	0	2	0
0	0	0	3
4	1	1	0

k=2

0	5	8	1
∞	0	3	∞
∞	∞	0	2
3	8	11	0

0	1	2	1
0	0	2	0
0	0	0	3
4	1	2	0

k=3

0	5	8	1
∞	0	3	5
∞	∞	0	2
3	8	11	0

0	1	2	1
0	0	2	3
0	0	0	3
4	1	2	0

k=4

0	5	8	1
8	0	3	5
5	10	0	2
3	8	11	0

0	1	2	1
4	0	2	3
4	1	0	3
4	1	2	0

Floyd–Warshall

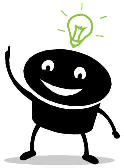
- Algoritmul funcționează corect chiar dacă arcele au și costuri negative (dar graful nu are circuite negative)

- Cum putem detecta pe parcursul algoritmului existența unui circuit negativ ($\Rightarrow$ datele de intrare nu sunt corecte) ?

Floyd–Warshall

- Algoritmul funcționează corect chiar dacă arcele au și costuri negative (dar graful nu are circuite negative)

- Cum putem detecta pe parcursul algoritmului existența unui circuit negativ ($\Rightarrow$ datele de intrare nu sunt corecte) ?



Există circuit negativ $\Leftrightarrow$ exista i cu $d_{ii} < 0$

Aplicație
Închiderea tranzitivă a unui graf orientat
Algoritmul Roy-Warhsall

Roy–Warshall

- **Aplicație:** **Închiderea tranzitivă** a unui graf orientat G=(V, E) (!!!neponderat):

  $G^* = (V, E^*)$, unde

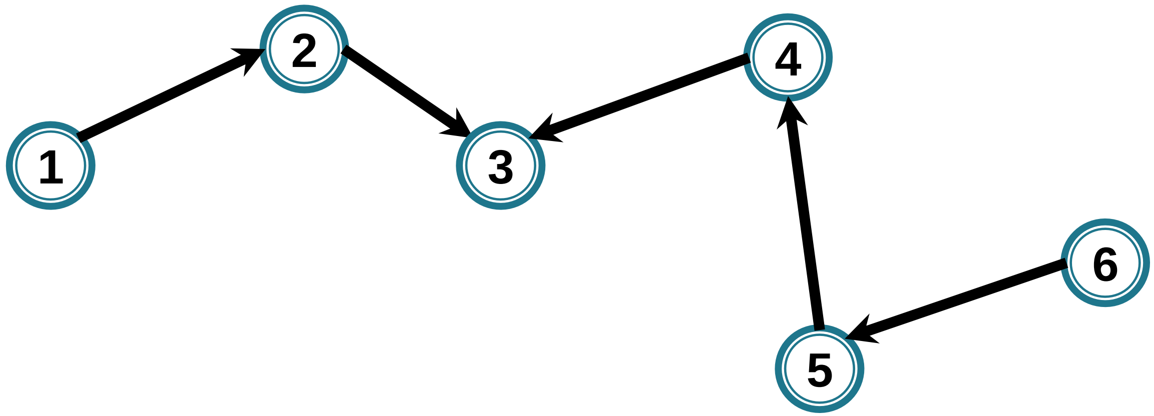
  $E^* = \{(i, j) \mid$ **există drum** (de lungime minim 1) de la i la j în G$\}$

  - **Utilitate:**

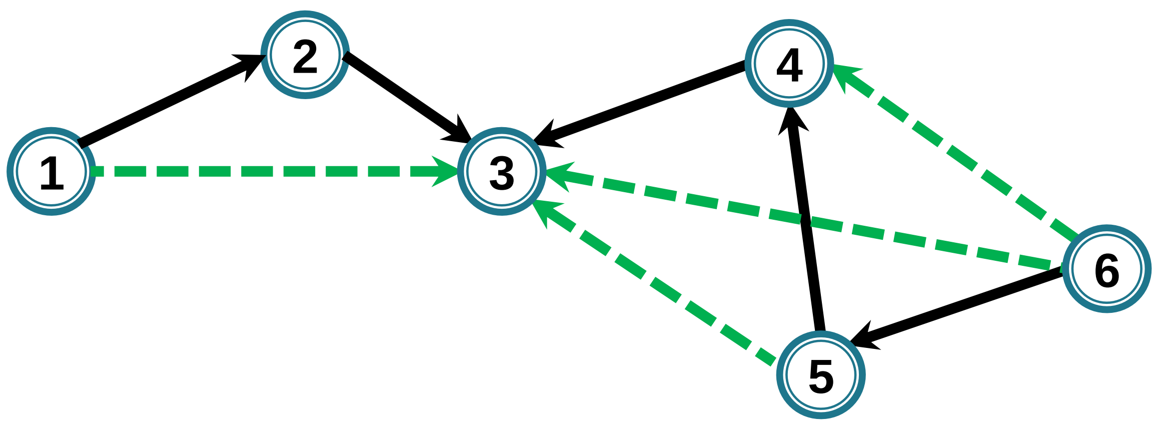
    – **grupări de obiecte aflate în relație (directă sau indirectă):** optimizări în baze de date, analize în rețele, logică

Roy-Warshall

Exemplul 1

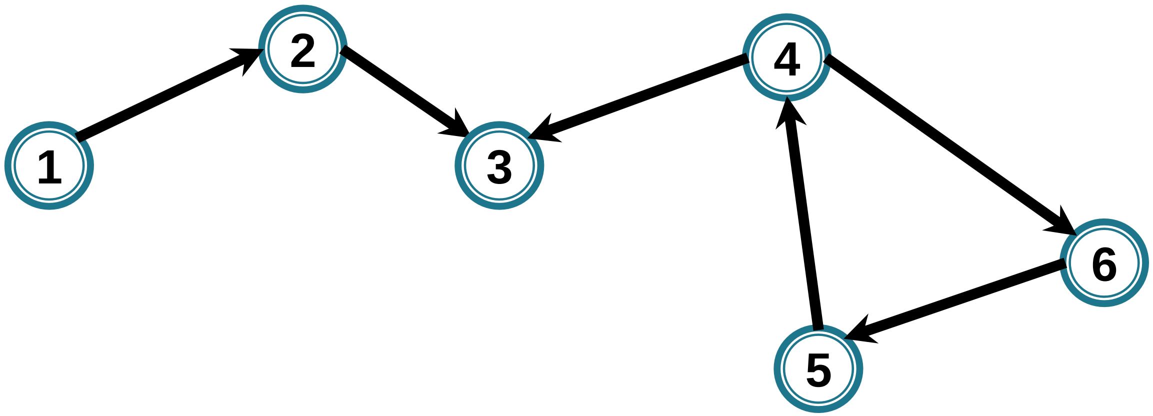


Închiderea tranzitivă

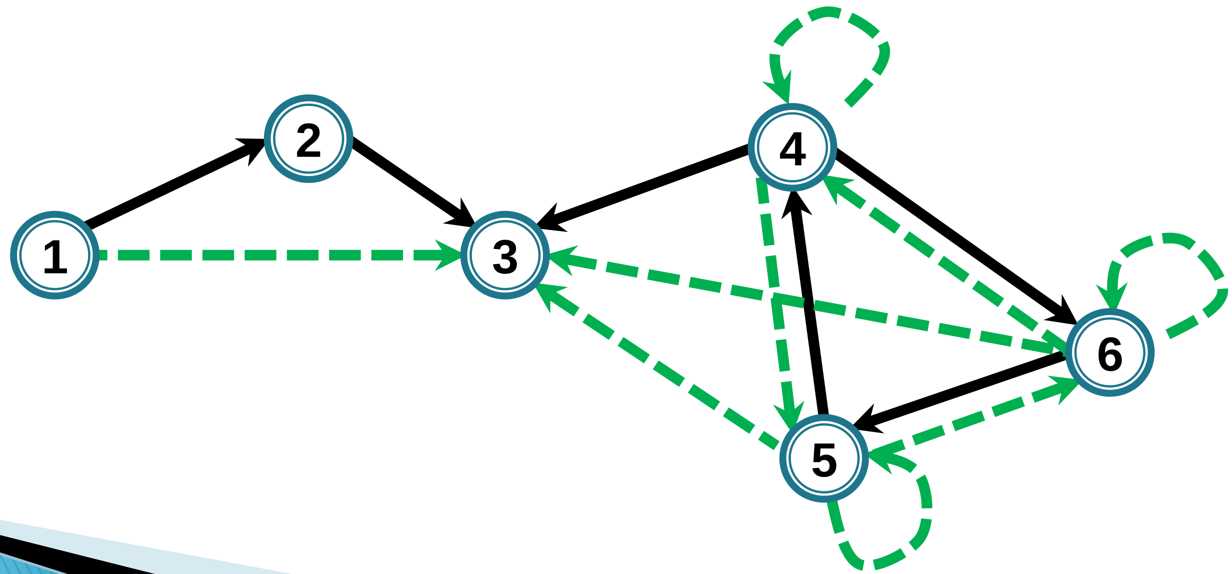


Roy–Warshall

Exemplul 2



Închiderea tranzitivă



Roy–Warshall

- **Închiderea tranzitivă** $\Leftrightarrow$ calculăm **matricea existenței drumurilor** (matricea de adiacență a închiderii tranzitive)

  $D = (d_{ij})_{i,j=1..n}$:

  d_{ij} = **1, dacă există drum nevid de la i la j**

  0, altfel

Roy–Warshall

```
initial d = matricea de adiacenta

for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
         d[i][j] = d[i][j] ∨ (d[i][k] ∧ d[k][j]);
                         SAU            SI
```

Roy-Warshall

- **Observație**

Dacă A este matricea de adiacență a unui graf și

$\mathbf{A^k} = (a^k_{ij})_{i,j=1..n}$: **puterea k a matricei** (k<n)

atunci $\mathbf{a^k_{ij}}$ = **numărul de drumuri distincte** de lungime k de la i la j (!nu neapărat elementare)

Demonstrație – Inducție (suplimentar)

Roy-Warshall

- **Observație**

  Dacă A este matricea de adiacență a unui graf și

  $\mathbf{A^k} = (a^k_{ij})_{i,j=1..n}$: **puterea k a matricei** (k<n)

  atunci $\mathbf{a^k_{ij}}$ = **numărul de drumuri distincte** de lungime k de la i la j (!nu neapărat elementare)

  - **Consecință**

    $$D = A \vee A^2 \vee \dots \vee A^{n-1}$$

  unde o valoare diferită de 0 se interpretează ca true