

Concluzii

Drumuri minime de sursă unică –
algoritmi

▶ Algoritmi – $G=(V, E)$ graf orientat

G – neponderat

Parcurgere lăţime BF

BF(s)

```
coada C ← ∅;  
adauga(s, C)
```

G – ponderat, ponderi >0

Algoritmul lui Dijkstra

Dijkstra(s)

```
(min-heap) Q ← V  
{se putea incepe doar cu Q ← {s}  
+vector viz;  $v \in Q \Leftrightarrow v$  nevizitat}
```

G – ponderat fără circuite

DAGS(s)

```
SortTop ← sortare_topologica(G)
```

► **Algoritmi** – $G=(V, E)$ graf orientat

G – neponderat

Parcurgere lățime BF

BF (s)

coada $C \leftarrow \emptyset;$

adauga (s, C)

pentru fiecare $u \in V$

```
d[u]=∞; tata[u]=viz[u]=0
```

```
viz[s] ← 1; d[s] ← 0
```

G – ponderat, ponderi >0

Algoritmul lui Dijkstra

Dijkstra (s)

```
(min-heap) Q ← v
```

{se putea incepe doar cu $Q \leftarrow \{s\}$ }

```
+vector viz; v ∈ Q ⇔ v nevizitat}
```

pentru fiecare $u \in V$

```
d[u] = ∞; tata[u]=0
```

$$d[s] = 0$$

G – ponderat fără circuite

DAGS (s)

```
SortTop ← sortare_topologica(G)
```

pentru fiecare $u \in V$

```
d[u] = ∞; tata[u]=0
```

$$d[s] = 0$$

▶ Algoritmi – $G=(V, E)$ graf orientat

G – neponderat	G – ponderat, ponderi >0	G – ponderat fără circuite
Parcurgere lăţime BF	Algoritmul lui Dijkstra	
BF (s)	Dijkstra (s)	DAGS (s)
<code>coada C ← ∅;</code> <code>adauga (s, C)</code>	<code>(min-heap) Q ← V</code> {se putea incepe doar cu <code>Q ← {s}</code> +vector viz; <code>v ∈ Q ⇔ v</code> nevizitat}	<code>SortTop ← sortare_topologica (G)</code>
pentru fiecare <code>u ∈ V</code> <code>d[u]=∞; tata[u]=viz[u]=0</code>	pentru fiecare <code>u ∈ V</code> <code>d[u] = ∞; tata[u]=0</code>	pentru fiecare <code>u ∈ V</code> <code>d[u] = ∞; tata[u]=0</code>
<code>viz[s]← 1; d[s] ← 0</code>	<code>d[s] = 0</code>	<code>d[s] = 0</code>
cat timp <code>C ≠ ∅</code> <code>u ← extrage (C);</code> pentru fiecare <code>uv ∈ E</code>	cat timp <code>Q ≠ ∅</code> <code>u = extrage (Q)</code> vârf cu eticheta <code>d</code> minimă pentru fiecare <code>uv ∈ E</code>	pentru fiecare <code>u ∈ SortTop</code> pentru fiecare <code>uv ∈ E</code>

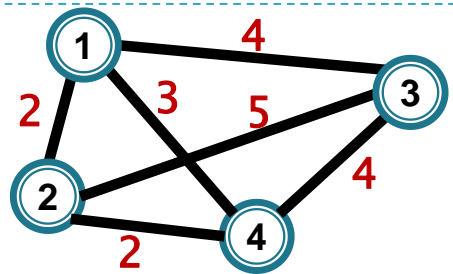
► Algoritmi – $G=(V, E)$ graf orientat

G – neponderat	G – ponderat, ponderi >0	G – ponderat fără circuite
Parcurgere lăţime BF	Algoritmul lui Dijkstra	
BF (s)	Dijkstra (s)	DAGS (s)
<code>coada C ← ∅;</code> <code>adauga (s, C)</code>	<code>(min-heap) Q ← V</code> {se putea incepe doar cu <code>Q ← {s}</code> <code>+vector viz; v ∈ Q ⇔ v nevizitat</code> }	<code>SortTop ← sortare_topologica(G)</code>
pentru fiecare $u \in V$ <code>d[u]=∞; tata[u]=viz[u]=0</code>	pentru fiecare $u \in V$ <code>d[u] = ∞; tata[u]=0</code>	pentru fiecare $u \in V$ <code>d[u] = ∞; tata[u]=0</code>
<code>viz[s]← 1; d[s] ← 0</code>	<code>d[s] = 0</code>	<code>d[s] = 0</code>
cat timp $C \neq \emptyset$ $u \leftarrow \text{extrage}(C);$ pentru fiecare $uv \in E$ daca <code>viz[v]=0</code> <code>d[v] ← d[v]+1</code> <code>tata[v] ← u</code> <code>adauga (v, C)</code> <code>viz[v] ← 1</code>	cat timp $Q \neq \emptyset$ $u = \text{extrage}(Q)$ vârf cu eticheta d minimă pentru fiecare $uv \in E$ daca $v \in Q$ si <code>d[u]+w(u,v)<d[v]</code> <code>d[v] = d[u]+w(u,v)</code> <code>tata[v] = u</code> <code>repara (v,Q)</code>	pentru fiecare $u \in \text{SortTop}$ pentru fiecare $uv \in E$ daca <code>d[u]+w(u,v)<d[v]</code> <code>d[v] = d[u]+w(u,v)</code> <code>tata[v] = u</code>
scrie d, tata	scrie d, tata	scrie d, tata

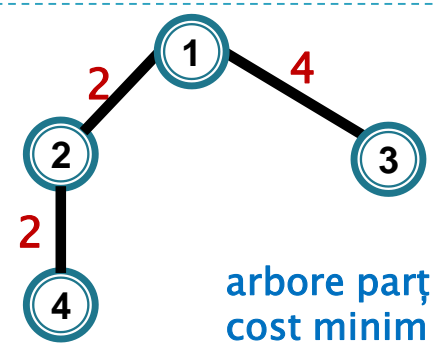
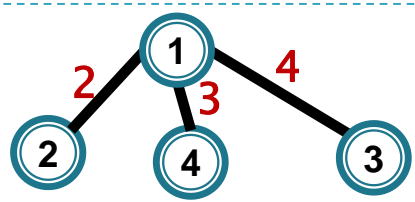
► Algoritmi – $G=(V, E)$ graf orientat

G – neponderat	G – ponderat, ponderi >0	G – ponderat fără circuite
Parcurgere lăţime BF	Algoritmul lui Dijkstra	
BF (s)	Dijkstra (s)	DAGS (s)
<code>coada C ← ∅;</code> <code>adauga (s, C)</code>	<code>(min-heap) Q ← V</code> {se putea incepe doar cu <code>Q ← {s}</code> +vector viz; <code>v ∈ Q ⇔ v</code> nevizitat}	<code>SortTop ← sortare_topologica(G)</code>
pentru fiecare <code>u ∈ V</code> <code>d[u]=∞; tata[u]=viz[u]=0</code>	pentru fiecare <code>u ∈ V</code> <code>d[u] = ∞; tata[u]=0</code>	pentru fiecare <code>u ∈ V</code> <code>d[u] = ∞; tata[u]=0</code>
<code>viz[s]← 1; d[s] ← 0</code>	<code>d[s] = 0</code>	<code>d[s] = 0</code>
cat timp <code>C ≠ ∅</code> <code>u ← extrage (C);</code> pentru fiecare <code>uv ∈ E</code> daca <code>viz[v]=0</code> <code>d[v] ← d[v]+1</code> <code>tata[v] ← u</code> <code>adauga (v, C)</code> <code>viz[v] ← 1</code>	cat timp <code>Q ≠ ∅</code> <code>u = extrage (Q)</code> vârf cu eticheta <code>d</code> minimă pentru fiecare <code>uv ∈ E</code> daca <code>v ∈ Q</code> si <code>d[u]+w(u,v)<d[v]</code> <code>d[v] = d[u]+w(u,v)</code> <code>tata[v] = u</code> <code>repara (v, Q)</code>	pentru fiecare <code>u ∈ SortTop</code> pentru fiecare <code>uv ∈ E</code> daca <code>d[u]+w(u,v)<d[v]</code> <code>d[v] = d[u]+w(u,v)</code> <code>tata[v] = u</code>
scrie <code>d, tata</code>	scrie <code>d, tata</code>	scrie <code>d, tata</code>
$O(n+m)$	$O(m \log(n)) / O(n^2)$	$O(n+m)$

Drumuri minime din s \Rightarrow arbore de drumuri minime (distanțe) din s
 \neq arbore parțial de cost minim – minimizează costul total



arbore al drumurilor minime față de 1

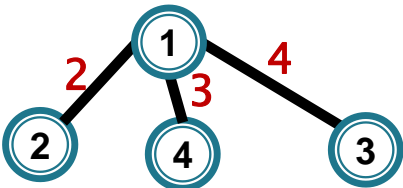
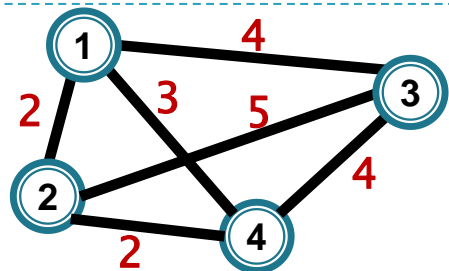


arbore parțial de cost minim

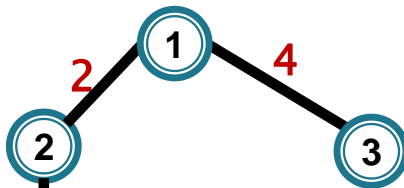
Drumuri minime din s \Rightarrow arbore de drumuri minime (distanțe) din s
 \neq arbore parțial de cost minim – minimizează costul total

G–(ne)**orientat** ponderat,
ponderi >0
Drumuri minime din s
Algoritmul lui Dijkstra

G– **neorientat** ponderat
ponderi reale
Arbore parțial de cost minim
Algoritmul lui Prim



arbore al drumurilor minime față de 1



arbore parțial de cost minim

Drumuri minime din s \Rightarrow arbore de drumuri minime (distanțe) din s

\neq arbore parțial de cost minim – minimizează costul total

G-(ne)orientat ponderat,
ponderi >0

Drumuri minime din s

Algoritmul lui Dijkstra

G- neorientat ponderat
ponderi reale

Arbore parțial de cost minim

Algoritmul lui Prim

Dijkstra(s)

(min-heap) Q \leftarrow V

pentru fiecare $u \in V$

$d[u] = \infty$; tata[u]=0

$d[s] = 0$

cat timp Q $\neq \emptyset$

u = extrage(Q) vârf cu eticheta

d minimă

pentru fiecare $uv \in E$

daca ~~v \in Q~~ si $d[u] + w(u,v) < d[v]$

$d[v] = d[u] + w(u,v)$

tata[v] = u

repara(v,Q)

scrie d, tata

Prim(s)

(min-heap) Q \leftarrow V

pentru fiecare $u \in V$

$d[u] = \infty$; tata[u]=0

$d[s] = 0$

cat timp Q $\neq \emptyset$

u = extrage(Q) vârf cu

eticheta d minimă

pentru fiecare $uv \in E$

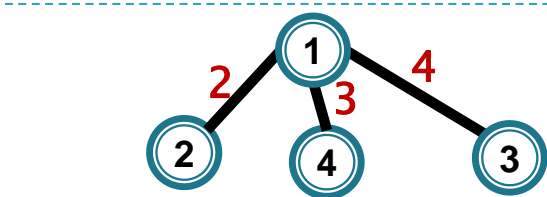
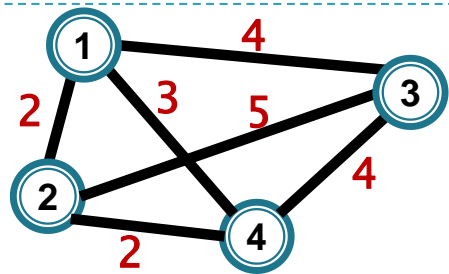
daca ~~v \in Q~~ si $w(u,v) < d[v]$

$d[v] = w(u,v)$

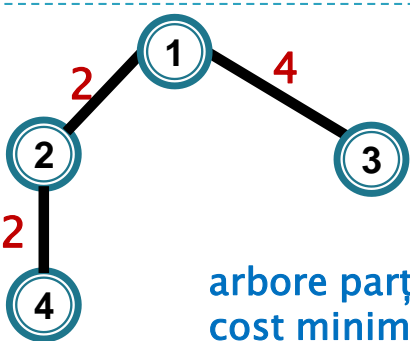
tata[v] = u

repara(v,Q)

scrie (u, tata[u]), pentru $u \neq s$



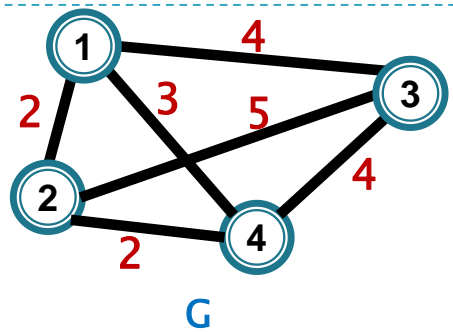
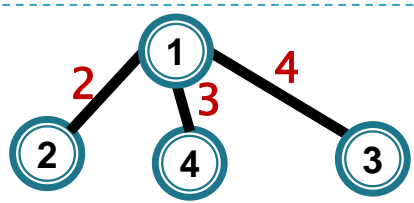
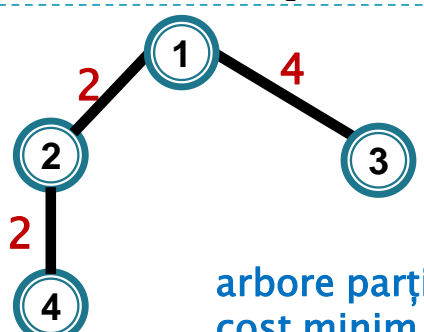
arbore al drumurilor minime față de 1



arbore parțial de
cost minim

Drumuri minime din s \Rightarrow arbore de drumuri minime (distanțe) din s

\neq arbore parțial de cost minim – minimizează costul total

	<div>G-(ne)orientat ponderat, ponderi >0</div> <div>Drumuri minime din s</div> <div>Algoritmul lui Dijkstra</div>	<div>G- neorientat ponderat ponderi reale</div> <div>Arbore parțial de cost minim</div> <div>Algoritmul lui Prim</div>
	<div>Dijkstra(s)</div> <div>(min-heap) Q \leftarrow V</div> <div>pentru fiecare u\inV</div> <div> d[u] = ∞; tata[u]=0</div> <div>d[s] = 0</div> <div>cat timp Q $\neq \emptyset$</div> <div> u = extrage(Q) vârf cu eticheta</div> <div> d minimă</div> <div> pentru fiecare uv\inE</div> <div> daca v\inQ si d[u]+w(u,v)<d[v]</div> <div> d[v] = d[u]+w(u,v)</div> <div> tata[v] = u</div> <div> repara(v,Q)</div> <div>scrie d, tata</div>	<div>Prim(s)</div> <div>(min-heap) Q \leftarrow V</div> <div>pentru fiecare u\inV</div> <div> d[u] = ∞; tata[u]=0</div> <div>d[s] = 0</div> <div>cat timp Q $\neq \emptyset$</div> <div> u = extrage(Q) vârf cu</div> <div> eticheta d minimă</div> <div> pentru fiecare uv\inE</div> <div> daca v\inQ si w(u,v)<d[v]</div> <div> d[v] = w(u,v)</div> <div> tata[v] = u</div> <div> repara(v,Q)</div> <div>scrie (u, tata[u]), pentru u\neqs</div>
 <div>G</div>	 <div>arbore al drumurilor minime față de 1</div>	 <div>arbore parțial de cost minim</div>

