Concluzii
Drumuri minime de sursă unică - algoritmi

Algoritmi – G=(V, E) graf orientat

G - neponderat
Parcurgere în lățime (BF)

BF(s)

```
coada C ← ∅
adauga(s, C)
```

G - ponderat, ponderi > 0
Algoritmul lui Dijkstra

Dijkstra(s)

```
(min-heap) Q ← V
// se putea începe doar cu Q ← {s} +
vector viz; v∈Q ⇔ v nevizitat
```

G - ponderat fără circuite

DAGS(s)

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

Algoritmi – G=(V, E) graf orientat

G - neponderat **Parcurgere în lățime (BF)**	G - ponderat, ponderi > 0 **Algoritmul lui Dijkstra**	G - ponderat fără circuite
BF(s)	**Dijkstra(s)**	**DAGS(s)**
coada C ← ∅ adauga(s, C)	(min-heap) Q ← V // se putea începe doar cu **Q ← {s}** + vector viz; **v∈Q ⇔** v nevizitat	SortTop ← sortare_topologica(G)
pentru fiecare u∈V d[u]=∞; tata[u]=viz[u]=0 viz[s]=1; d[s]=0	**pentru** fiecare u∈V d[u]=∞; tata[u]=0 d[s] = 0	**pentru** fiecare u∈V d[u]=∞; tata[u]=0 d[s] = 0

Algoritmi – G=(V, E) graf orientat

G - neponderat
Parcurgere în lățime (BF)

BF(s)

```
coada C ← ∅
adauga(s, C)

pentru fiecare u∈V
    d[u]=∞; tata[u]=viz[u]=0

viz[s]=1; d[s]=0

cât timp C ≠ ∅
    u ← extrage(C)

    pentru fiecare uv∈E
```

G - ponderat, ponderi > 0
Algoritmul lui Dijkstra

Dijkstra(s)

```
(min-heap) Q ← V
// se putea începe doar cu Q ← {s} +
vector viz; v∈Q ⇔ v nevizitat

pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0

cât timp Q ≠ ∅
    u = extrage(Q) vârf cu
        eticheta d minimă
    pentru fiecare uv∈E
```

G - ponderat fără circuite

DAGS(s)

```
SortTop ← sortare_topologica(G)

pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0

pentru fiecare u∈SortTop


    pentru fiecare uv∈E
```

Algoritmi – G=(V, E) graf orientat

G - neponderat
Parcurgere în lățime (BF)

BF(s)

```
coada C ← ∅
adauga(s, C)

pentru fiecare u∈V
    d[u]=∞; tata[u]=viz[u]=0

viz[s]=1; d[s]=0

cât timp C ≠ ∅
    u ← extrage(C)

    pentru fiecare uv∈E
        dacă viz[v] = 0
            d[v] = d[v] + 1
            tata[v]=u
            adauga(v, C)
            viz[v]=1

scrie d, tata
```

G - ponderat, ponderi > 0
Algoritmul lui Dijkstra

Dijkstra(s)

```
(min-heap) Q ← V
// se putea începe doar cu Q ← {s} +
vector viz; v∈Q ⇔ v nevizitat

pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0

cât timp Q ≠ ∅
    u = extrage(Q) vârf cu
        eticheta d minimă
    pentru fiecare uv∈E
        dacă v∈Q și
d[u]+w(u,v)<d[v]
            d[v] = d[u] + w(u,v)
            tata[v] = u
            repara(v, Q)

scrie d, tata
```

G - ponderat fără circuite

DAGS(s)

```
SortTop ← sortare_topologica(G)

pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0

pentru fiecare u∈SortTop


    pentru fiecare uv∈E
        dacă d[u]+w(u,v) < d[v]
            d[v] = d[u] + w(u,v)
            tata[v] = u


scrie d, tata
```

Algoritmi – G=(V, E) graf orientat

G - neponderat
Parcurgere în lățime (BF)

BF(s)

```
coada C ← ∅
adauga(s, C)
```

```
pentru fiecare u∈V
    d[u]=∞; tata[u]=viz[u]=0

viz[s]=1; d[s]=0
```

```
cât timp C ≠ ∅
    u ← extrage(C)

    pentru fiecare uv∈E
        dacă viz[v] = 0
            d[v] = d[v] + 1
            tata[v]=u
            adauga(v, C)
            viz[v]=1
```

```
scrie d, tata
```

O(n+m)

G - ponderat, ponderi > 0
Algoritmul lui Dijkstra

Dijkstra(s)

```
(min-heap) Q ← V
// se putea începe doar cu Q ← {s} +
vector viz; v∈Q ⇔ v nevizitat
```

```
pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0
```

```
cât timp Q ≠ ∅
    u = extrage(Q) vârf cu
        eticheta d minimă
    pentru fiecare uv∈E
        dacă ~~v∈Q~~ și
d[u]+w(u,v)<d[v]
            d[v] = d[u] + w(u,v)
            tata[v] = u
            repara(v, Q)
```

```
scrie d, tata
```

O(m log n) / O(n²) / O(n log n + m)

G - ponderat fără circuite

DAGS(s)

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

```
pentru fiecare u∈V
    d[u]=∞; tata[u]=0

d[s] = 0
```

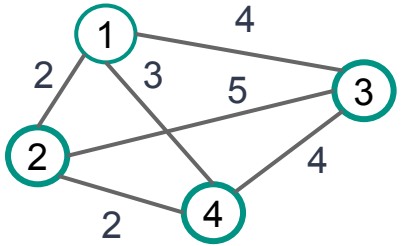
```
pentru fiecare u∈SortTop


    pentru fiecare uv∈E
        dacă d[u]+w(u,v) < d[v]
            d[v] = d[u] + w(u,v)
            tata[v] = u
```

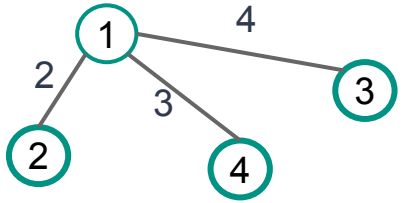
```
scrie d, tata
```

O(n+m)

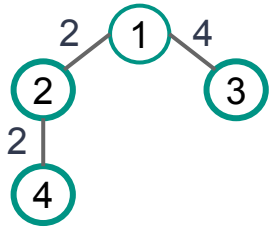
Drumuri minime vs APCM

Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s ≠ APCM - minimizează costul total





arbore al drumurilor minime față de 1



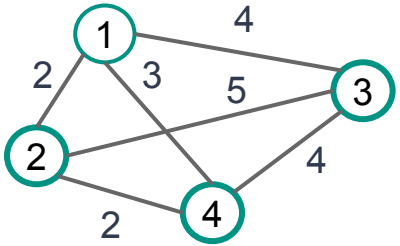
arbore parțial de cost minim

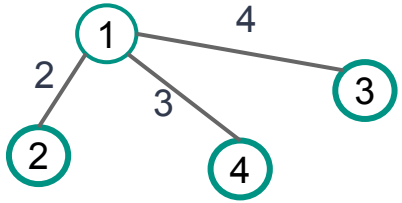
Drumuri minime vs APCM

Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s ≠ APCM - minimizează costul total

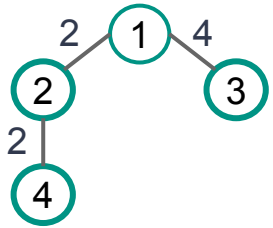
G - (ne)orientat ponderat, ponderi > 0
Drumuri minime din s
Algoritmul lui Dijkstra

G - neorientat ponderat, ponderi reale
Arbore parțial de cost minim
Algoritmul lui Prim





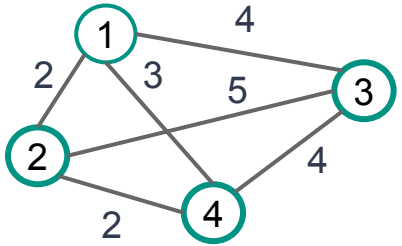
arbore al drumurilor minime față de 1



arbore parțial de cost minim

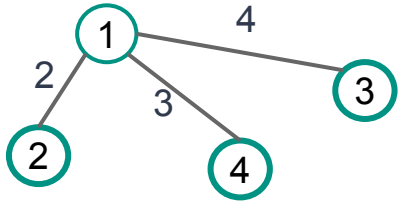
Drumuri minime vs APCM

Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s ≠ APCM - minimizează costul total



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

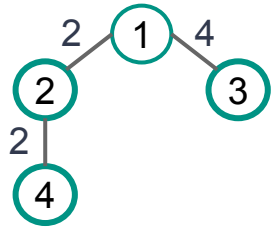
```
Dijkstra(s)
(min-heap) Q ← V
pentru fiecare u∈V
    d[u]=∞; tata[u]=0
d[s] = 0
cât timp Q ≠ ∅
    u = extrage(Q) vârf cu
        eticheta d minimă
    pentru fiecare uv∈E
        dacă ~~v∈Q~~ și d[u]+w(u,v)<d[v]
            d[v] = d[u] + w(u,v)
            tata[v] = u
            repara(v, Q)
scrie d, tata
```



arbore al drumurilor minime față de 1

G - neorientat ponderat, ponderi reale
Arbore parțial de cost minim
Algoritmul lui Prim

```
Prim(s)
(min-heap) Q ← V
pentru fiecare u∈V
    d[u]=∞; tata[u]=0
d[s] = 0
cât timp Q ≠ ∅
    u = extrage(Q) vârf cu
        eticheta d minimă
    pentru fiecare uv∈E
        dacă v∈Q și w(u,v)<d[v]
            d[v] = w(u,v)
            tata[v] = u
            repara(v, Q)
scrie d, tata, pentru u≠s
```



arbore parțial de cost minim