Concluzii

Drumuri minime de sursă unică – algoritmi

Algoritmi – G=(V, E) graf orientat

G – neponderat

Parcurgere lățime 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∈Q ⇔ 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 ← ∅;
adauga(s, C)

pentru fiecare u∈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 ← {s}
+vector viz; v∈Q ⇔ v nevizitat}

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

d[s] = 0
```

G – ponderat fără circuite

```
DAGS(s)

SortTop ← sortare_topologica(G)

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

d[s] = 0
```

Algoritmi – G=(V, E) graf orientat

G – neponderat
Parcurgere 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

cat timp C ≠ ∅
   u ←  extrage(C);

   pentru fiecare uv∈E
```

G – ponderat, ponderi >0
Algoritmul lui Dijkstra

```
Dijkstra(s)

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

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

d[s] = 0

cat 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 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

cat timp C ≠ ∅
   u ←  extrage(C);

   pentru fiecare uv∈E
       daca viz[v]=0
          d[v] ← d[u]+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 incepe doar cu Q ← {s}
+vector viz; v∈Q ⇔ v nevizitat}

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

d[s] = 0

cat timp Q ≠ ∅
   u = extrage(Q) vârf cu eticheta
        d minimă

   pentru fiecare uv∈E
      daca v∈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
```

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
        daca 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 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

cat timp C ≠ ∅
   u ←  extrage(C);

   pentru fiecare uv∈E
       daca viz[v]=0
          d[v] ← d[u]+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 incepe doar cu Q ← {s}
+vector viz; v∈Q ⇔ v nevizitat}

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

d[s] = 0

cat timp Q ≠ ∅
   u = extrage(Q) vârf cu eticheta
          d minimă

   pentru fiecare uv∈E
      daca v∈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
```

O(m log(n))/ O(n^2)

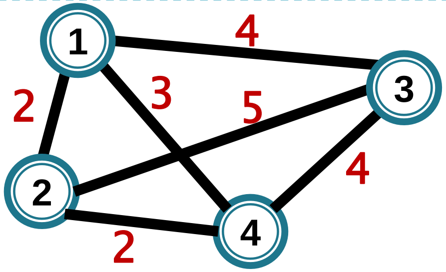
G – ponderat fără circuite

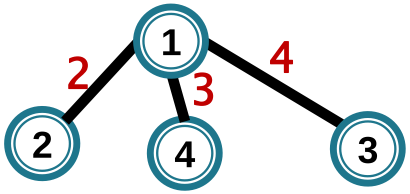
```
DAGS(s)

SortTop ← sortare_topologica(G)

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

d[s] = 0

pentru fiecare u ∈ SortTop

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

scrie d, tata
```

O(n+m)

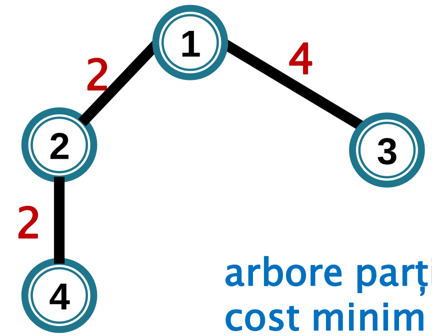
Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s

≠ **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

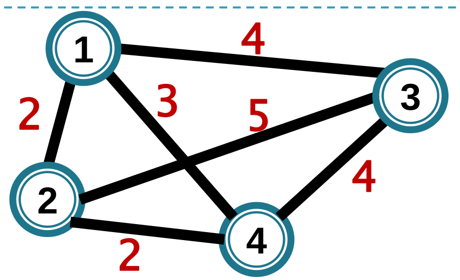
Drumuri minime din s

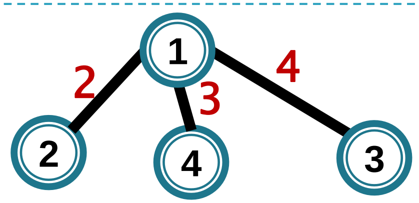
Algoritmul lui Dijkstra

G- neorientat ponderat
ponderi reale

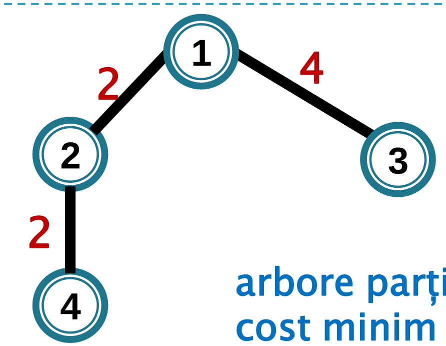
Arbore parțial de cost minim

Algoritmul lui Prim





arbore al drumurilor minime față de 1



arbore parțial de cost minim

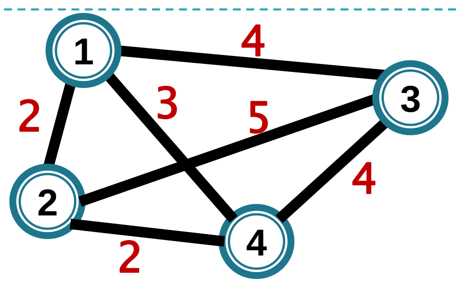
Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s

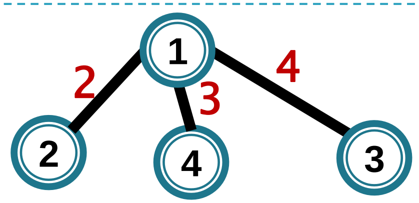
≠ arbore parțial de cost minim – minimizează costul total

G–(ne)orientat ponderat, ponderi >0	G– neorientat ponderat ponderi reale
Drumuri minime din s	Arbore parțial de cost minim
Algoritmul lui Dijkstra	Algoritmul lui Prim

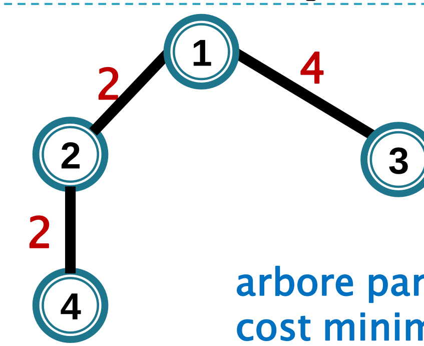
```
Dijkstra(s)
(min-heap) Q ← V
pentru fiecare u∈V
    d[u] = ∞; tata[u]=0
d[s] = 0
cat timp Q ≠ ∅
   u = extrage(Q) vârf cu eticheta
       d minimă
   pentru fiecare uv∈E
       daca v∈Q si d[u]+w(u,v)<d[v]
              d[v] = d[u]+w(u,v)
              tata[v] = u
              repara(v,Q)
scrie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

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





arbore al drumurilor minime față de 1



arbore parțial de cost minim

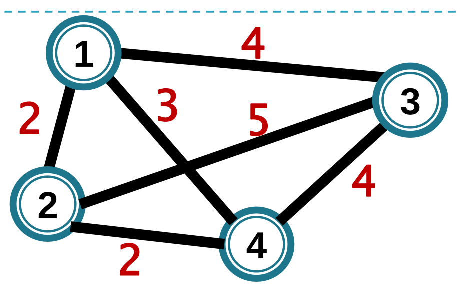
Drumuri minime din s ⇒ arbore de drumuri minime (distanțe) din s

≠ arbore parțial de cost minim – minimizează costul total

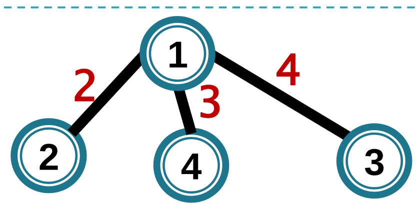
G–(ne)orientat ponderat, ponderi >0	G– neorientat ponderat ponderi reale
Drumuri minime din s	Arbore parțial de cost minim
Algoritmul lui Dijkstra	Algoritmul lui Prim

```
Dijkstra(s)
(min-heap) Q ← V
pentru fiecare u∈V
    d[u] = ∞; tata[u]=0
d[s] = 0
cat timp Q ≠ ∅
   u = extrage(Q) vârf cu eticheta
      d minimă
   pentru fiecare uv∈E
      daca v∈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
```

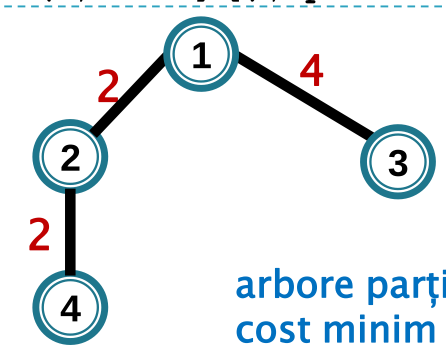
```
Prim(s)
(min-heap) Q ← V
pentru fiecare u∈V
    d[u] = ∞; tata[u]=0
d[s] = 0
cat timp Q ≠ ∅
   u = extrage(Q) vârf cu
      eticheta d minimă
   pentru fiecare uv∈E
      daca v∈Q si w(u,v)<d[v]
             d[v] = w(u,v)
             tata[v] = u
             repara(v,Q)
scrie (u, tata[u]), pentru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



G



arbore al drumurilor minime față de 1



arbore parțial de cost minim