Drumuri minime în grafuri ponderate

Aplicații

- Dată o hartă rutieră, să se determine
  - un drum minim între două orașe date
  - câte un drum minim între oricare două orașe de pe hartă

Aplicaţii

- **Determinarea de drumuri minime/distanţe – numeroase aplicaţii**
  - probleme de rutare
  - robotică
  - procesarea imaginilor
  - strategii jocuri
  - probleme de planificare (drumuri critice)

Cadru

Fie:

- G – graf orientat ponderat
- P – drum

$$w(P) = \sum_{e \in E(P)} w(e)$$

costul/ponderea/lungimea drumului P

Cadru

Fie:

- G – graf orientat ponderat
- Presupunem că G nu conține circuite de pondere negativă

Cadru

- Fie s,v$\in$V, **s $\neq$ v.**
- **Distanţa de la s la v**

  $\delta_G(s, v) = \min\{ w(P) \mid P$ este drum de la s la v$\}$

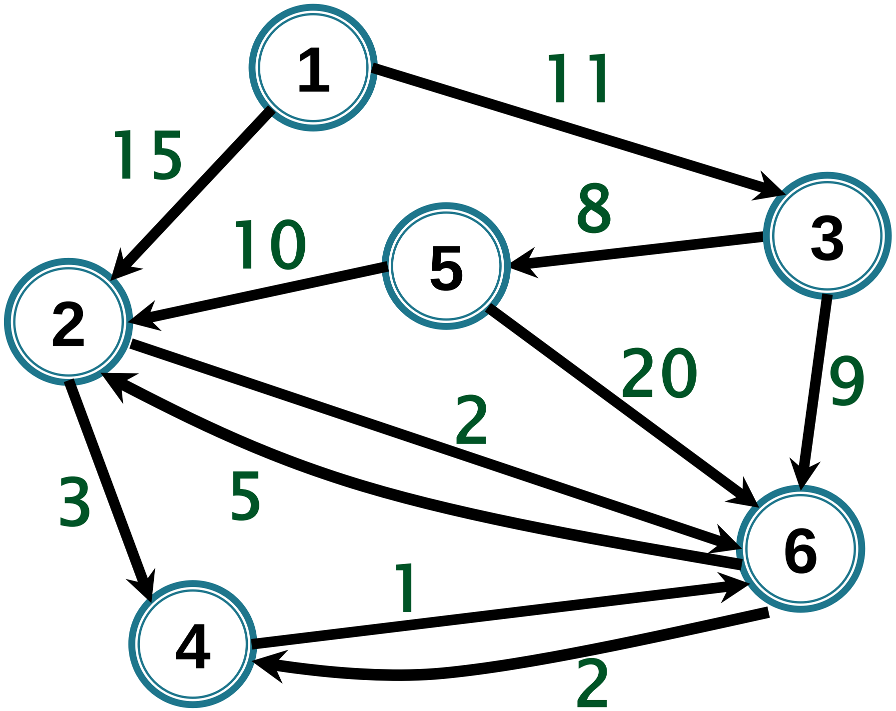
Cadru

- Fie $s, v \in V$, $s \neq v$.
- **Distanţa de la s la v**

  $$\delta_G(s, v) = \min\{ w(P) \mid P \text{ este drum de la } s \text{ la } v\}$$

  - $\delta_G(s, s) = 0$
  - **Convenție.** $\min \varnothing = \infty$

- Un drum P de la s la v se numește **drum minim de la s la v** dacă $w(P) = \delta_G(s, v)$



1
2
3
4
5
6
15
11
8
10
20
9
2
3
5
1
2

Tipuri de probleme de drumuri minime

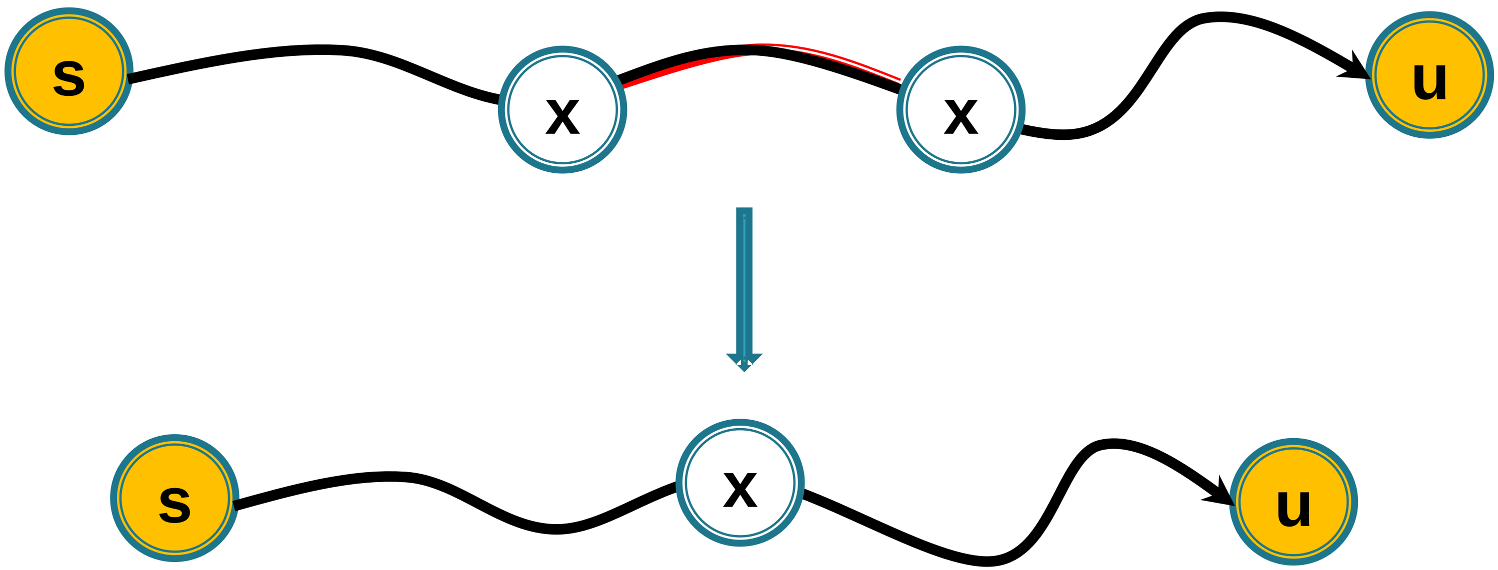
- între două vârfuri date
- de la un vârf la toate celelalte
- între oricare două vârfuri

Situaţii:

- Sunt acceptate şi arce de cost negativ?
- Graful conţine circuite?
- Graful conţine circuite de cost negativ?

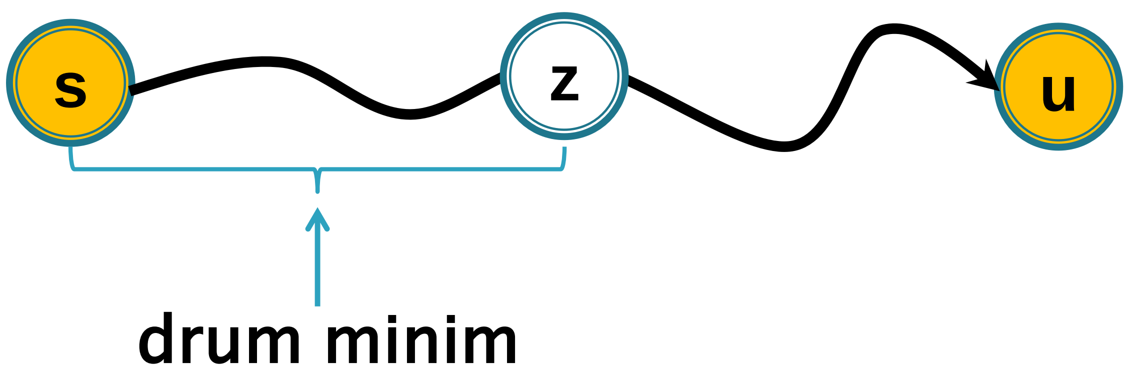
Observații

- **Observația 1.** Dacă P este un drum minim de la s la u și **nu există circuite de cost negativ**, atunci P este drum **elementar.**

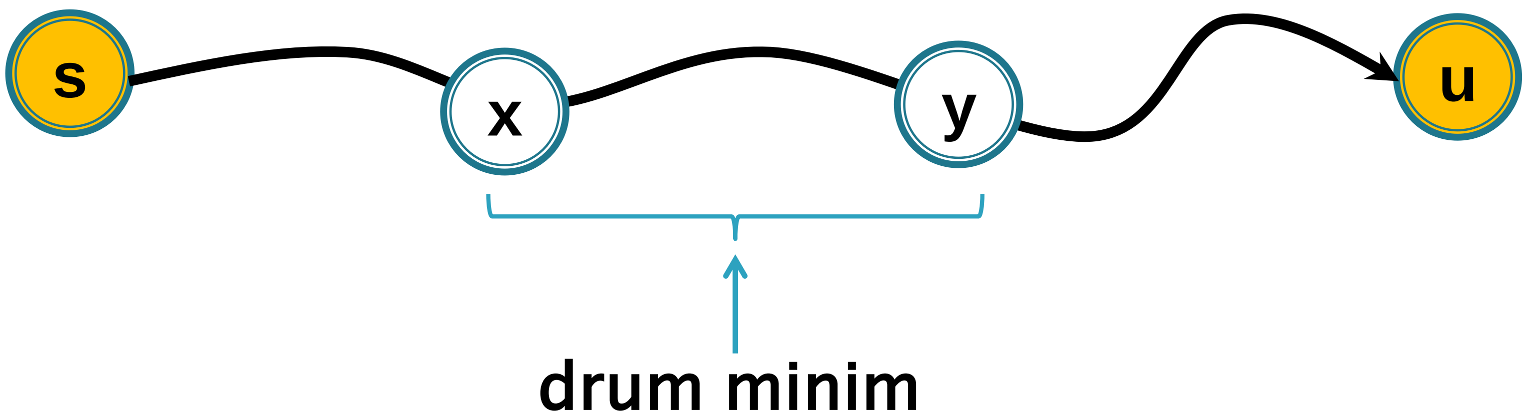


Observații

- **Observația 2.** Dacă P este un drum minim de la s la u și z este un vârf al lui P, atunci subdrumul lui P de la s la z este drum minim de la s la z.



Observații



Drumuri minime de la un vârf s dat la celelalte vârfuri (de sursă unică)

Problema drumurilor minime de sursă unică (de la s la celelalte vârfuri)

Se dau:

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

  $w : E \rightarrow \mathbb{R}$

- un vârf de start s

Să se determine distanța de la s la fiecare vârf x al lui G / la un vârf dat t (și un arbore al distanțelor față de s/ un drum minim de la s la t)

Drumuri minime de sursă unică s

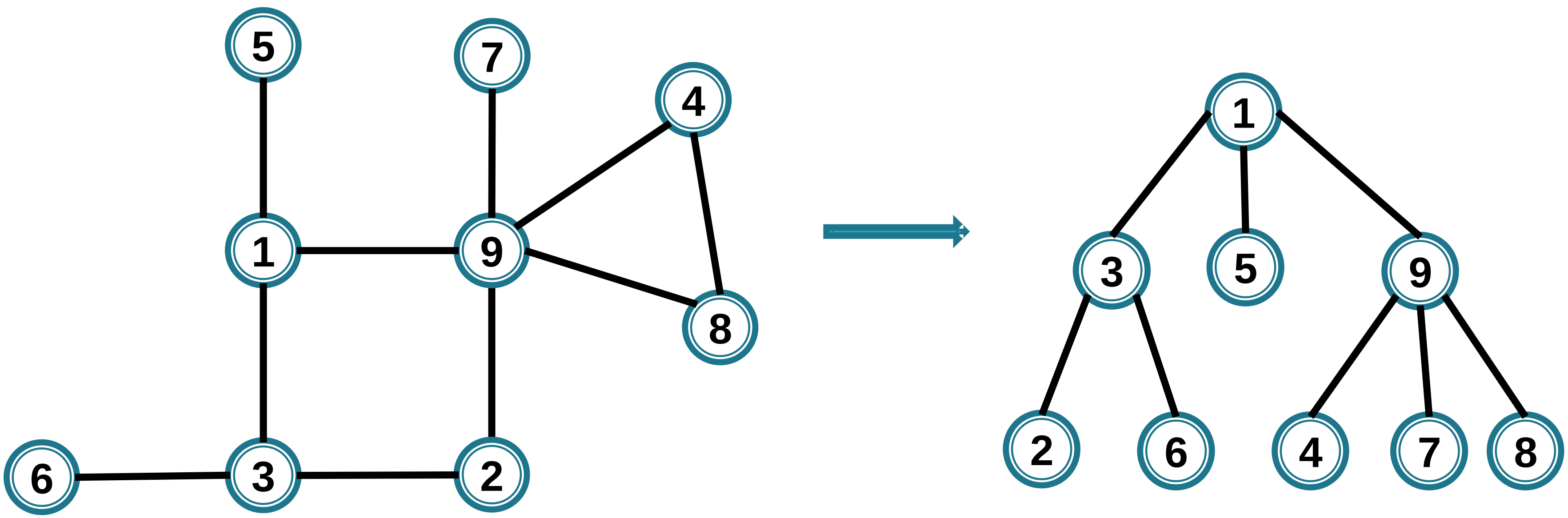
- Dacă G nu este ponderat, cum putem calcula distanţele faţă de s?

Drumuri minime de sursă unică s

- **Amintim**

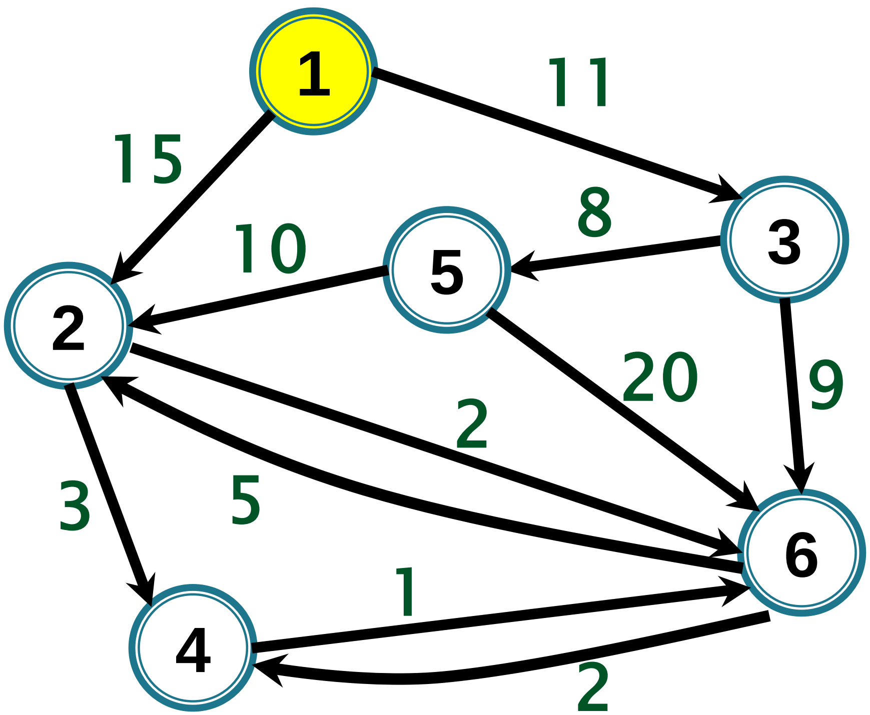
În cazul unui graf neponderat, problema se rezolvă folosind parcurgerea BF din s

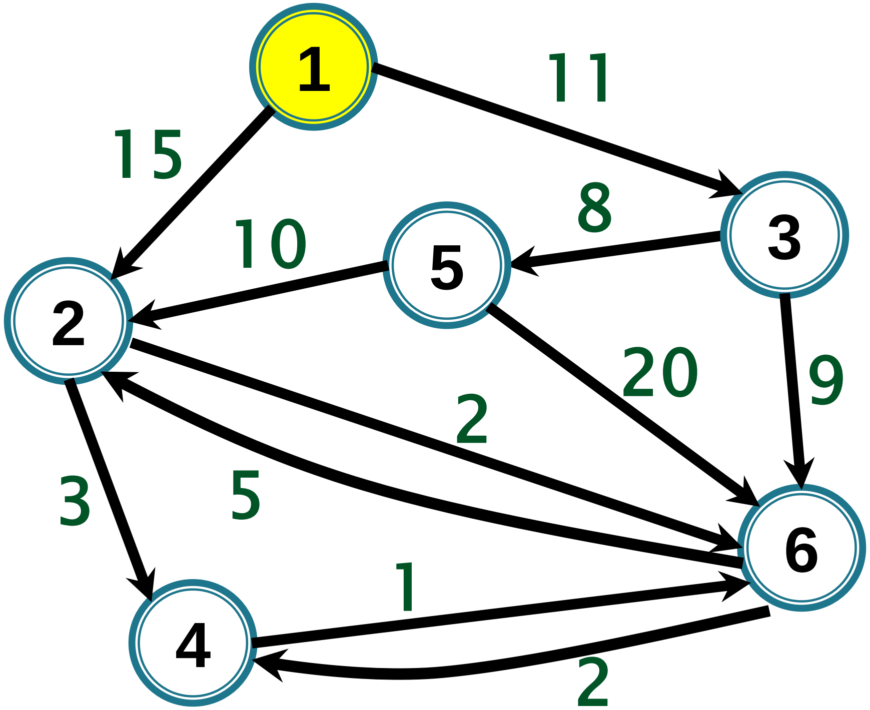
$\Rightarrow$ arborele BF (al distanțelor față de s)

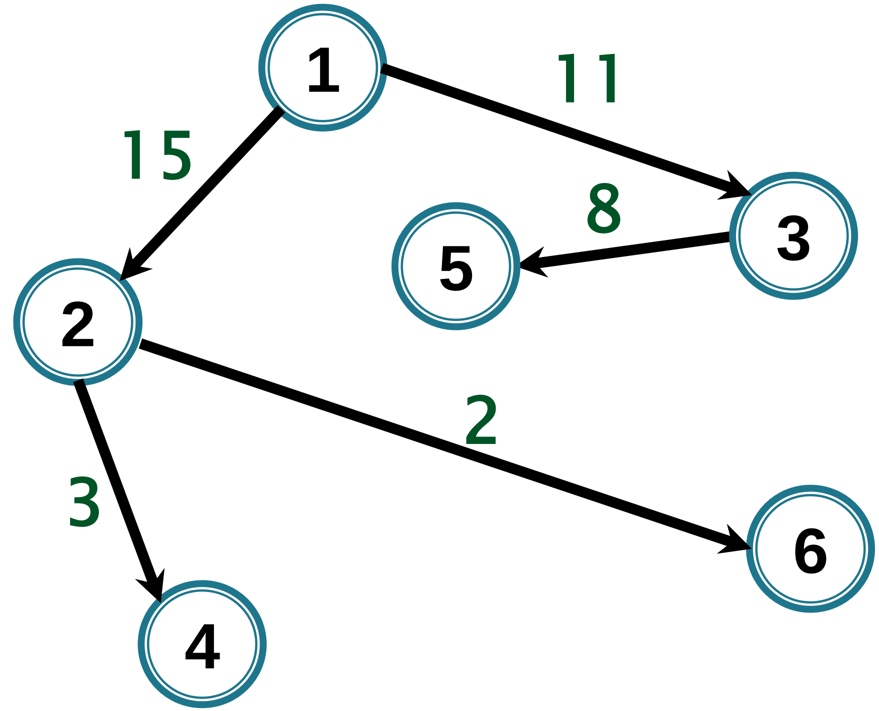


arborele BF

s = 1





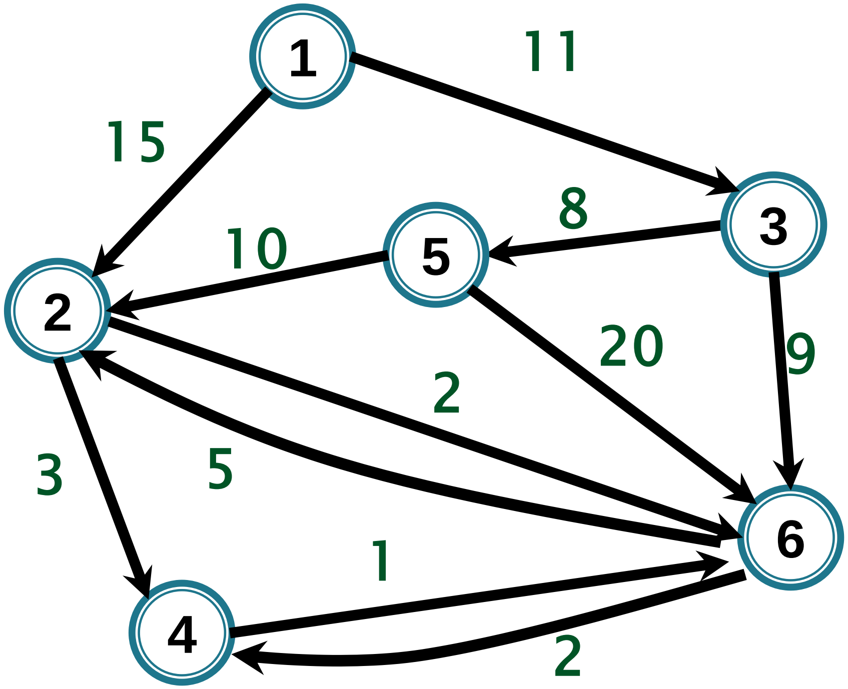


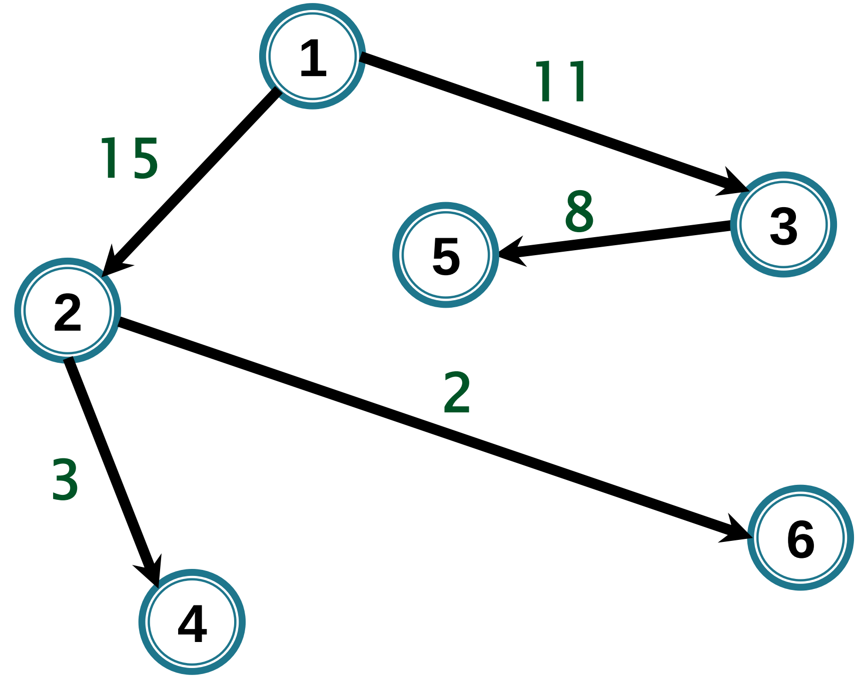
arbore al distanțelor față de 1

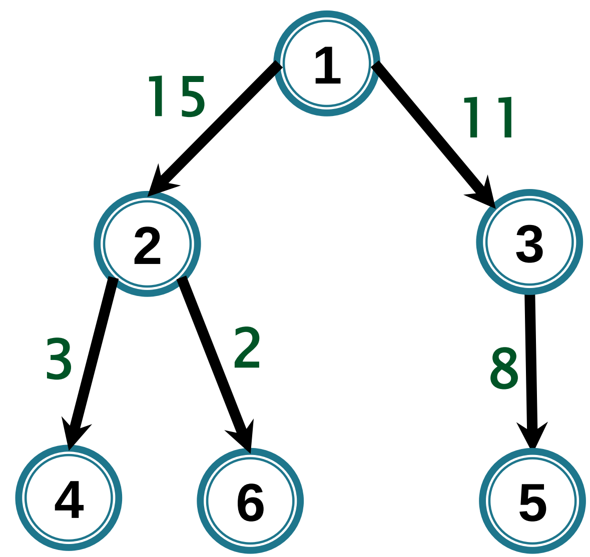
Definiție: Pentru un vârf dat s un arbore al distanțelor față de s = un subgraf T al lui G care **conservă distanțele** de la s la celelalte vârfuri accesibile din s

$$\delta_T(s, v) = \delta_G(s, v),\ \ \forall\ v \in V \text{ accesibil din } s,$$

graful neorientat asociat lui T fiind arbore cu rădăcina în s (cu arcele corespunzătoare orientate de la s la frunze)



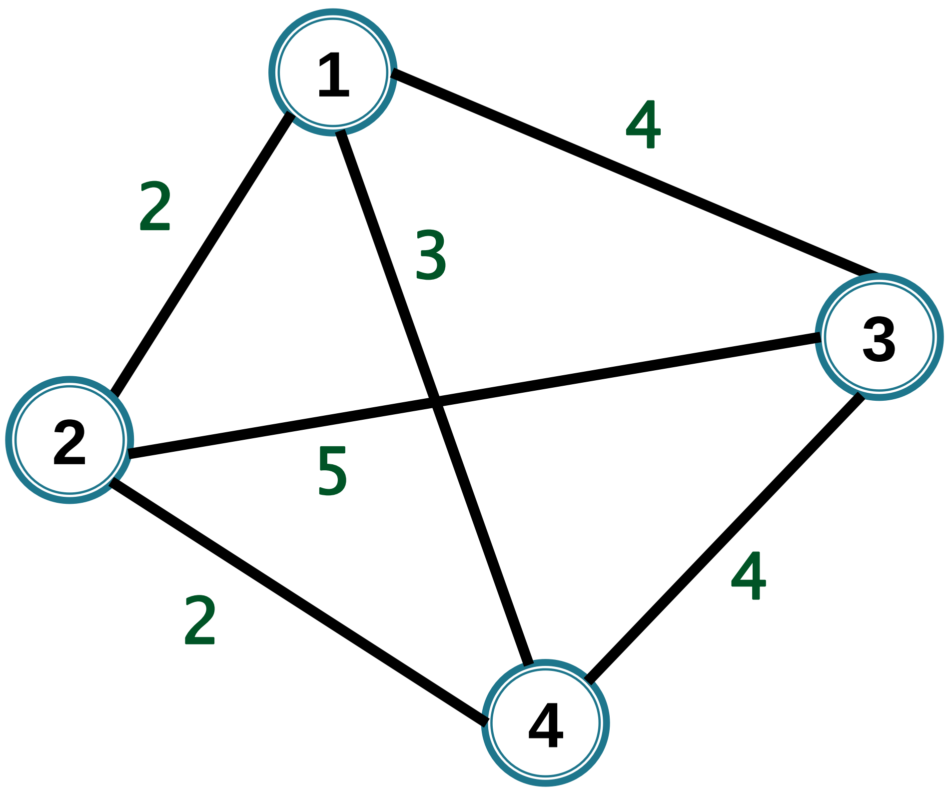




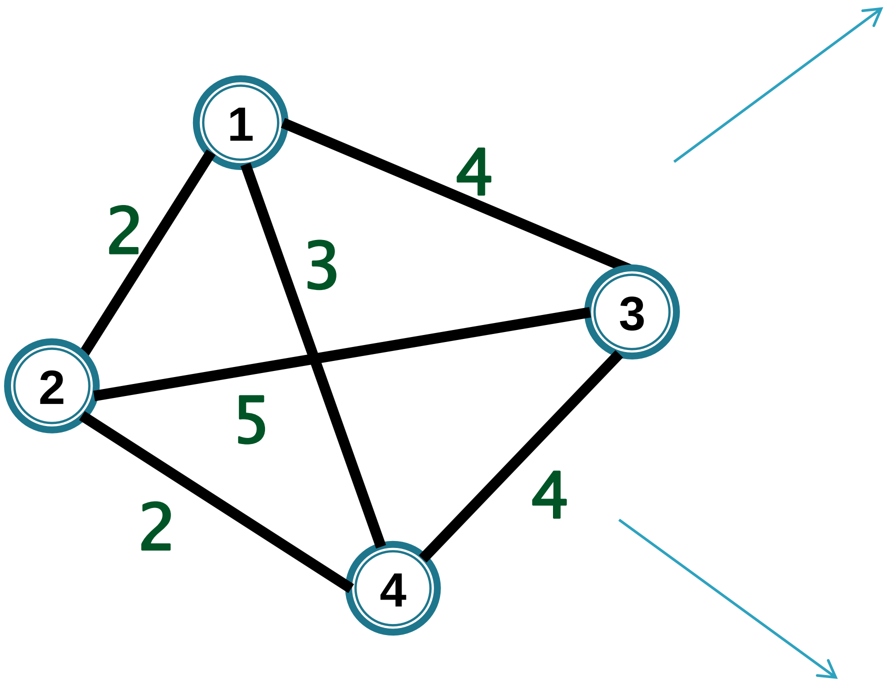
arbore al distanţelor faţă de 1

- Presupunem că toate vârfurile sunt accesibile din s
- Problema drumurilor minime de sursă unică este echivalentă cu determinarea unui arbore al distanțelor față de s

- Un arbore parțial de cost minim <u>nu</u> este neapărat un arbore de distanțe minime



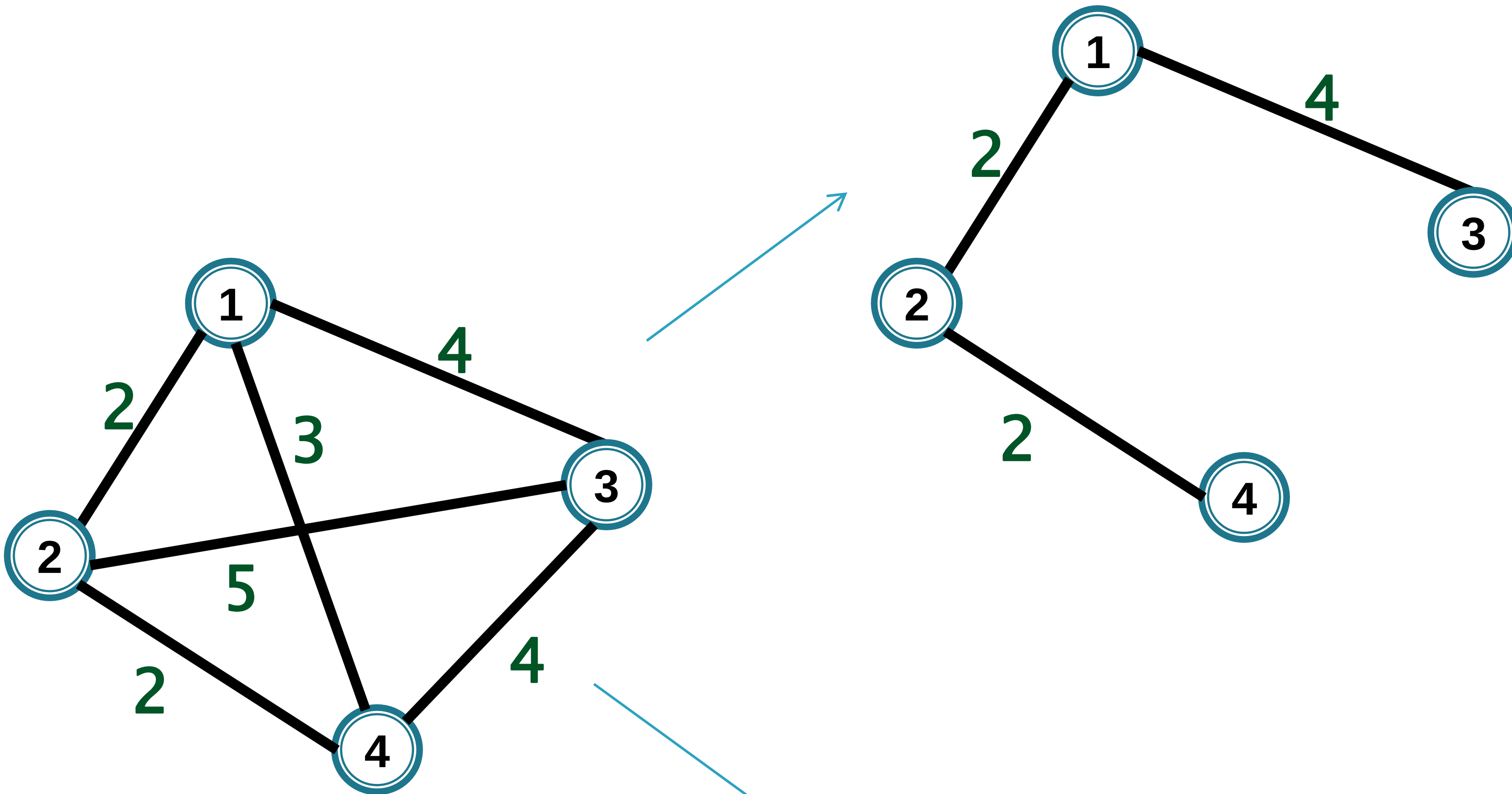
Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime



Apcm

Arbore al distanțelor față de 1

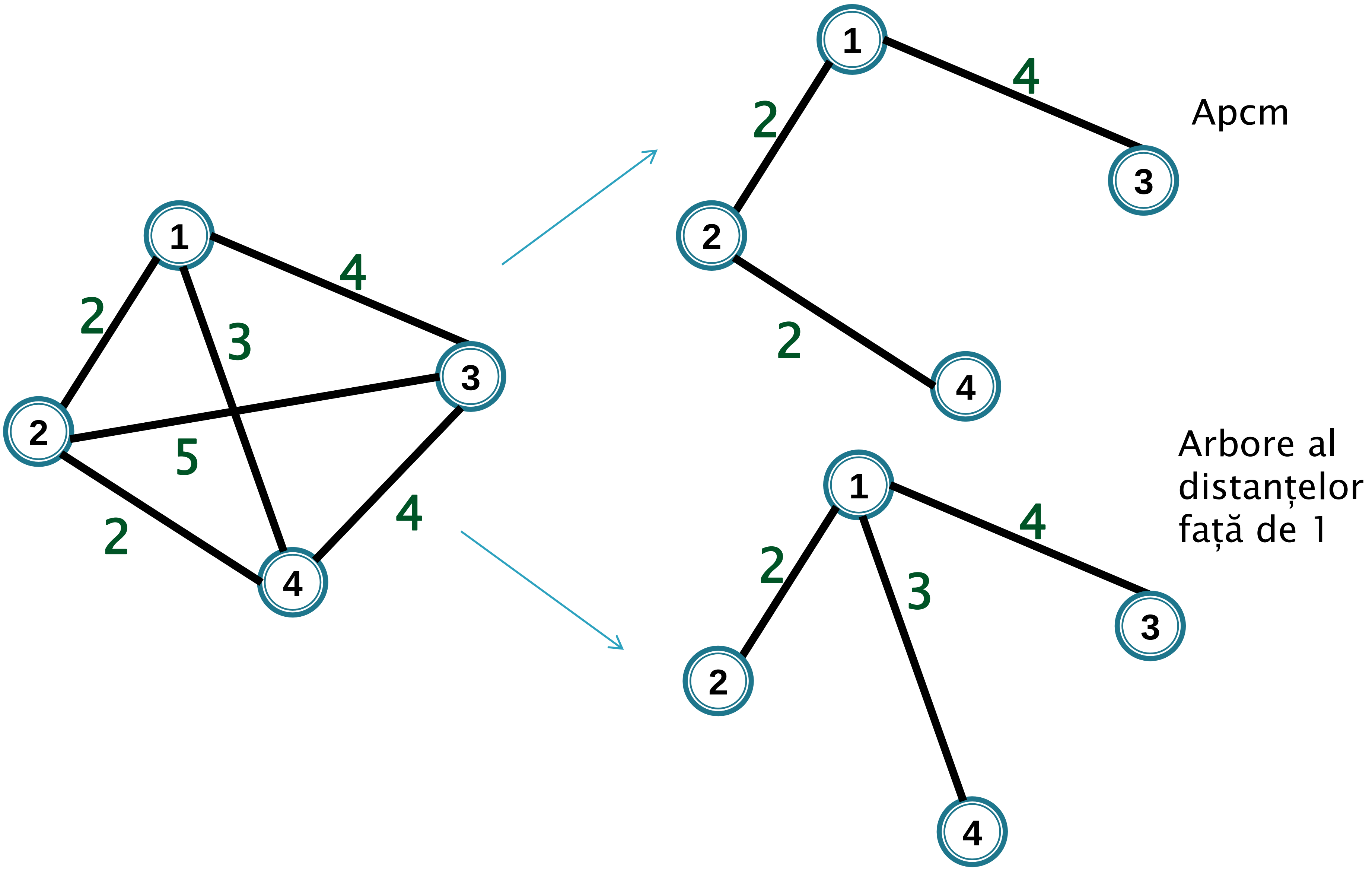
Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime



Apcm

Arbore al distanțelor față de 1

Un arbore parțial de cost minim nu este neapărat un arbore de distanțe minime

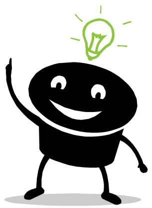


Drumuri minime de sursă unică s

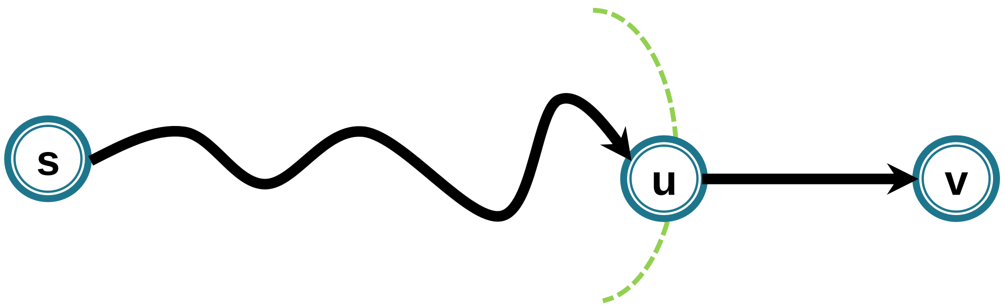
- În ce ordine considerăm vârfurile pentru a calcula distanţele față de s?

Drumuri minime de sursă unică s

- În ce ordine considerăm vârfurile pentru a calcula distanţele faţă de s?



"din aproape în aproape"



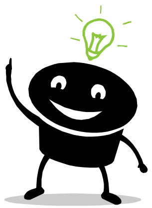
Dacă u este predecesor al lui v pe un drum minim de la s la v $\Rightarrow$

$\delta(s,v) = \delta(s,u) + w(uv)$

Ştim $\delta(s, u)$ $\Rightarrow$ aflăm şi $\delta(s, v)$

Drumuri minime de sursă unică s

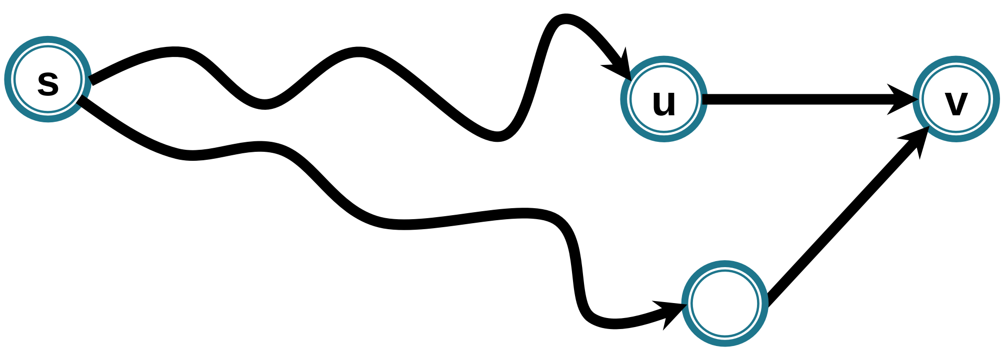
- În ce ordine considerăm vârfurile pentru a calcula distanţele faţă de s?



"din aproape în aproape" $\Rightarrow$

când considerăm un vârf v, pentru a calcula $\delta(s,v)$ ar fi util să ştim deja $\delta(s,u)$ pentru u cu $uv \in E$ (?!toate)

$$\delta(s,v) = \min\{ \delta(s,u) + w(u,v) \mid uv \in E \}$$

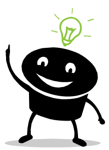


Drumuri minime de sursă unică s

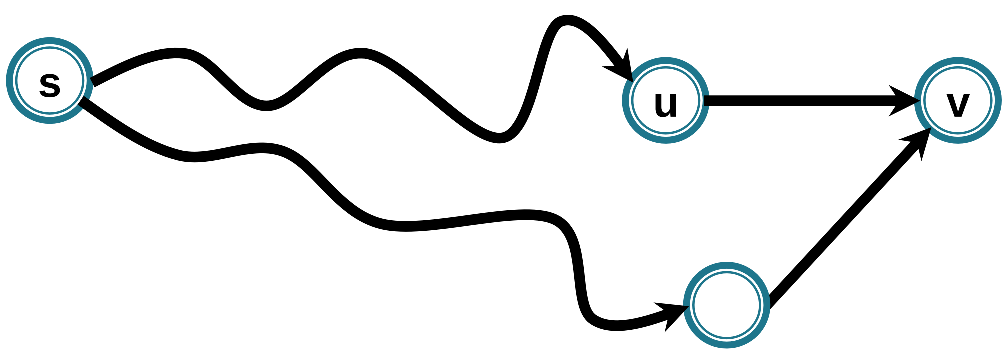
- În ce ordine considerăm vârfurile pentru a calcula distanţele față de s?

  “din aproape în aproape” $\Rightarrow$

  când considerăm un vârf v, pentru a calcula $\delta(s,v)$ ar fi util să ştim deja $\delta(s,u)$ pentru orice u cu $uv \in E$



Ar fi utilă o ordonare a vârfurilor astfel încât dacă $uv \in E$, atunci u se află înaintea lui v



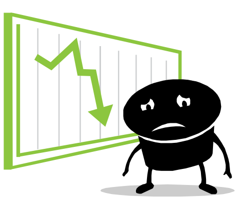
Drumuri minime de sursă unică s

- În ce ordine considerăm vârfurile pentru a calcula distanţele față de s?

  “din aproape în aproape” $\Rightarrow$

  când considerăm un vârf v, pentru a calcula $\delta(s,v)$ ar fi util să știm deja $\delta(s,u)$ pentru orice u cu $uv \in E$

  - Ar fi utilă o ordonare a vârfurilor astfel încât dacă $uv \in E$, atunci u se află înaintea lui v



O astfel de ordonare <u>nu</u> <u>există</u> dacă graful conţine circuite

Drumuri minime de sursă unică s

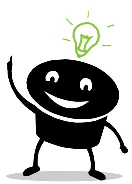
- În ce ordine considerăm vârfurile pentru a calcula distanţele faţă de s?

  “din aproape în aproape” $\Rightarrow$

  când considerăm un vârf v, pentru a calcula $\delta(s,v)$ ar fi util să ştim deja $\delta(s,u)$ pentru orice u cu $uv \in E$

  - Ar fi utilă o ordonare a vârfurilor astfel încât dacă $uv \in E$, atunci u se află înaintea lui v

  O astfel de ordonare <u>nu</u> <u>există</u> dacă graful conţine circuite



Dacă există circuite – <u>estimăm</u> distanţele pe parcursul algoritmului şi considerăm vârful care este **estimat** a fi cel mai aproape de s

Drumuri minime de sursă unică s

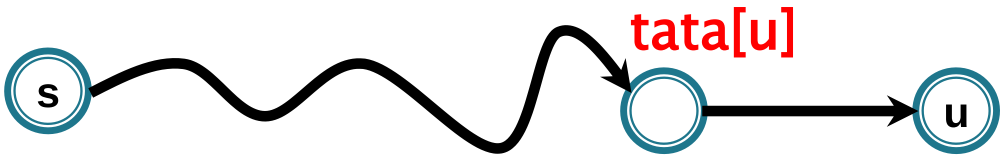
- Algoritmi pentru grafuri orientate cu circuite, dar cu ponderi pozitive – Dijkstra

- Algoritmi pentru grafuri orientate fără circuite (cu ponderi reale) DAGs = Directed Acyclic Graphs

- Algoritmi pentru grafuri orientate cu circuite și ponderi reale, care detectează existența de circuite negative – Bellman-Ford

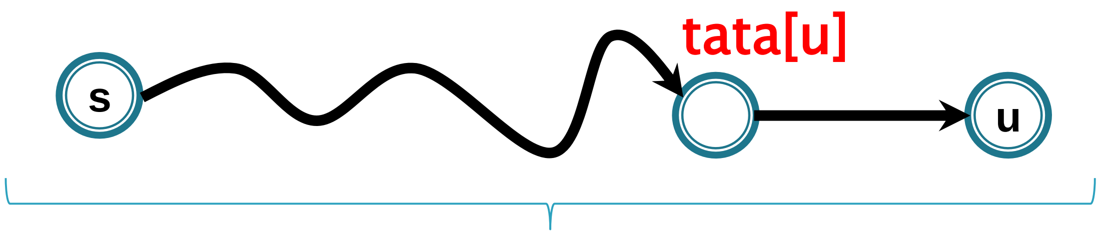
Drumuri minime de sursă unică s

- **Idei comune:** Pe parcursul algoritmului fiecare vârf are asociate informațiile:
  - **d[u]** – etichetă de distanță
  - **tata[u]**



Drumuri minime de sursă unică s

- **Idei comune:** **Pe parcursul algoritmului fiecare vârf are asociate informațiile:**
  - **d[u]** – etichetă de distanță
  - **tata[u]**

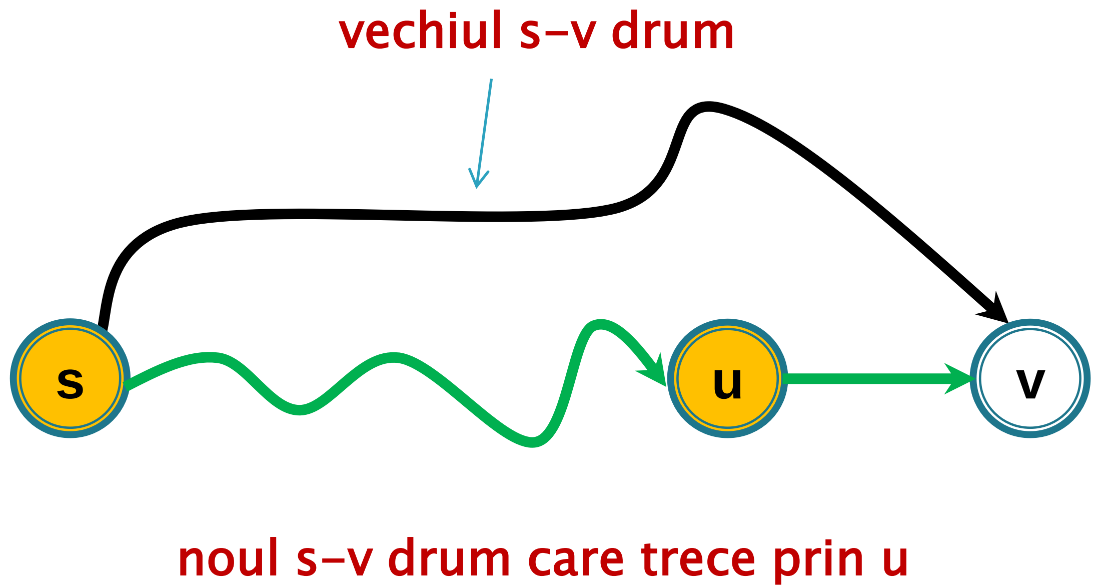


d[u] = costul minim al unui drum de la s la u descoperit până la acel moment

tata[u] = **predecesorul** lui u pe drumul de cost minim de la s la u descoperit până la acel moment

Drumuri minime de sursă unică s

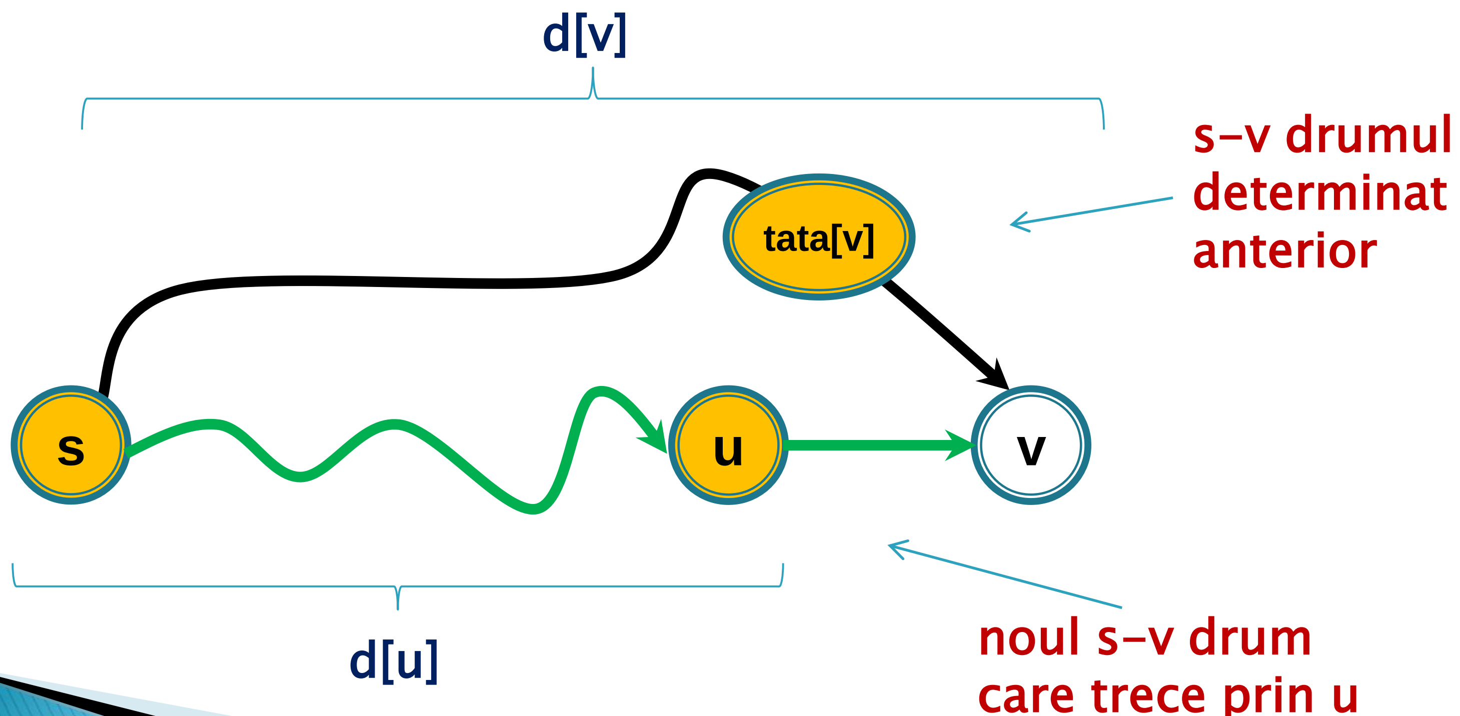
- **Relaxarea unui arc (u, v)** = a verifica dacă d[v] poate fi îmbunătăţit trecând prin vârful u



Drumuri minime de sursă unică s

- Relaxarea unui arc (u, v) :

```
dacă d[u] + w(u,v) < d[v] atunci
     d[v] = d[u] + w(u,v);
     tata[v] = u
```

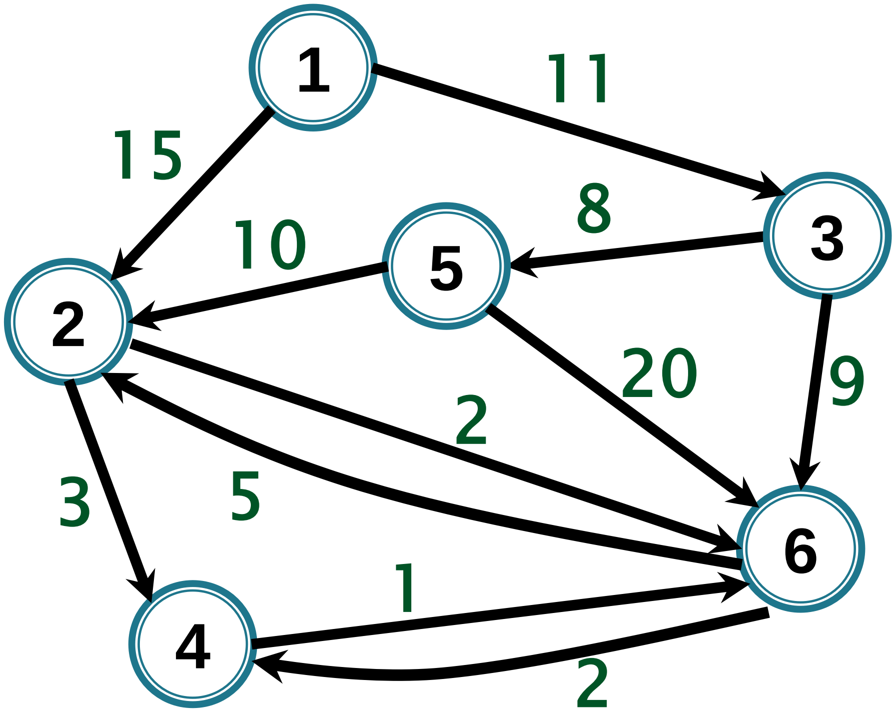


Algoritmul lui Dijkstra

- **Ipoteză:**

  Presupunem că arcele au <u>cost pozitiv</u>

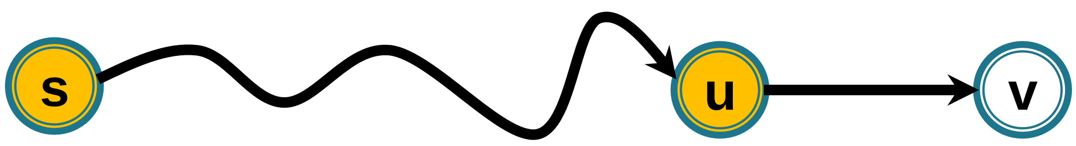
  (graful poate conţine circuite)


1
2
3
4
5
6
11
15
8
10
20
9
2
3
5
1
2

Algoritmul lui Dijkstra

Idee: La un pas este ales ca vârf curent (vizitat) vârful u care <u>estimat</u> a fi cel mai apropiat de s

- Estimarea pentru u = cel mai scurt drum de la s la u determinat până la pasul curent



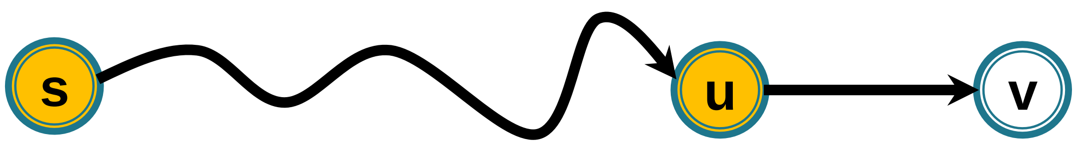
Algoritmul lui Dijkstra

Idee: La un pas este ales ca vârf curent (vizitat) vârful u care <u>estimat</u> a fi cel mai apropiat de s

- Estimarea pentru u = cel mai scurt drum de la s la u determinat până la pasul curent

+ se descoperă noi drumuri către vecinii lui $\Rightarrow$

se actualizează distanțele estimate pentru vecini



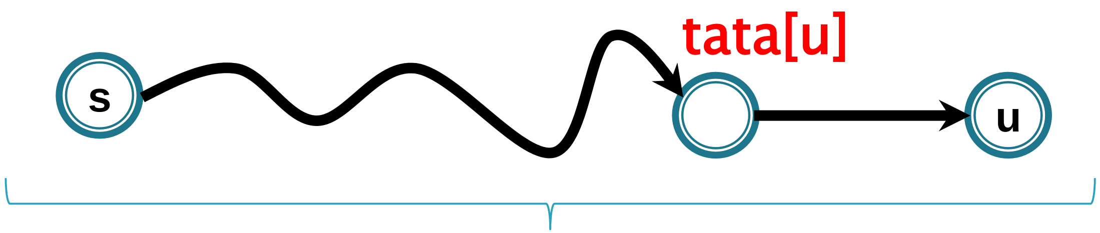
Algoritmul lui Dijkstra

- generalizare a ideii de parcurgere BF
- dacă toate arcele au cost egal Dijkstra $\equiv$ BF

Pseudocod

Algoritmul lui Dijkstra

- Reţinem pentru fiecare vârf etichetele
  - d[u]
  - tata[u]



d[u] = costul minim al unui drum de la s la u descoperit până la acel moment

Algoritmul lui Dijkstra

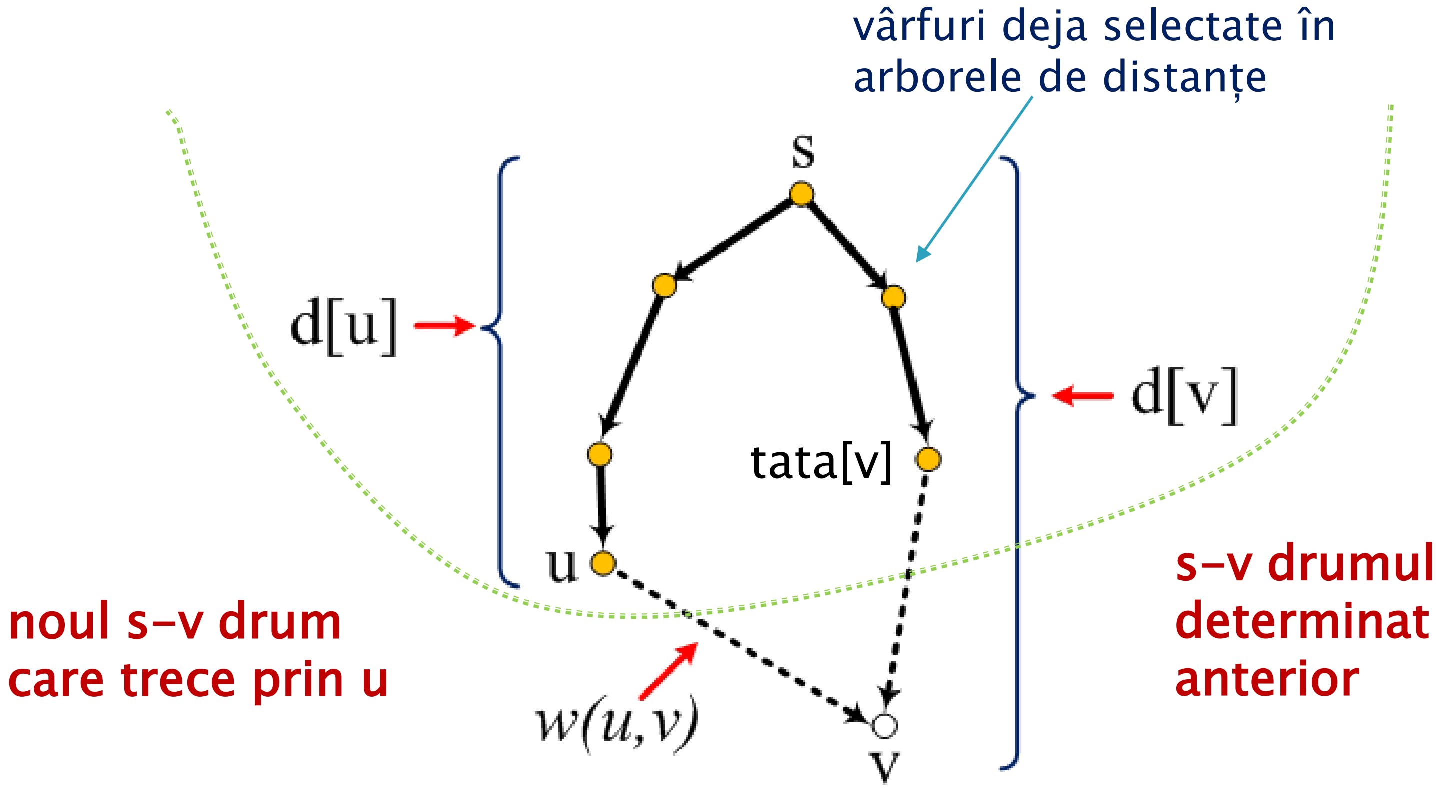
- **La un pas**
  - este selectat un vârf u (neselectat) care “**pare**” cel mai apropiat de s $\Leftrightarrow$ are eticheta d minimă

Algoritmul lui Dijkstra

- **La un pas**
  - este selectat un vârf u (neselectat) care "**pare**" cel mai apropiat de s ⇔ are eticheta d minimă
  - Se actualizează etichetele d[v] ale vecinilor lui u – considerând drumuri care trec prin u
    - **tehnica de relaxare a arcelor care ies din u**

Algoritmul lui Dijkstra

Raportat la vârfuri deja selectate - similar Prim



Relaxarea unui arc (u, v) :

```
dacă d[u] + w(u,v) < d[v] atunci
      d[v] = d[u] + w(u,v);
     tata[v] = u
```

Dijkstra(G, w, s)

```
  inițializează mulțimea vârfurilor neselectate Q cu V
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
     d[u] = ∞; tata[u]=0
```

Dijkstra(G, w, s)

```
  inițializează mulțimea vârfurilor neselectate Q cu V
  pentru fiecare u∈V executa
      d[u] = ∞; tata[u]=0
   d[s] = 0
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa ⇔ pentru i=1,n executa
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
       u = extrage vârf cu eticheta d minimă din Q
```

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
       u = extrage vârf cu eticheta d minimă din Q
       pentru fiecare uv∈E executa  //relaxare uv
```

Dijkstra(G, w, s)

```
inițializează mulțimea vârfurilor neselectate Q cu V
pentru fiecare u∈V executa
       d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
       u = extrage vârf cu eticheta d minimă din Q
       pentru fiecare uv∈E executa
            daca v∈Q si d[u]+w(u,v)<d[v] atunci
                   d[v] = d[u]+w(u,v)
                   tata[v] = u
```

Dijkstra(G, w, s)

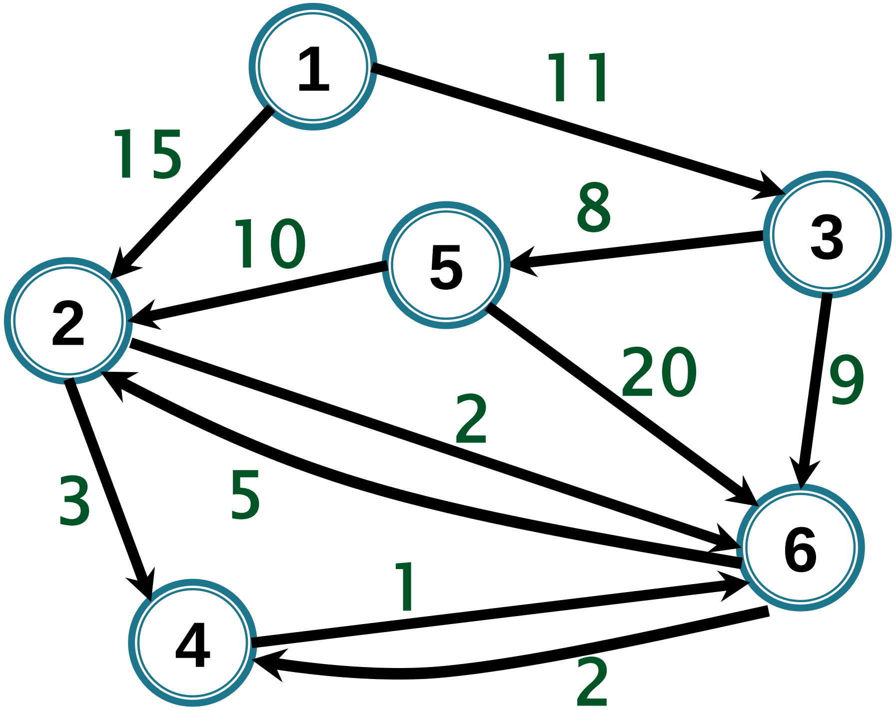
```
 inițializează mulțimea vârfurilor neselectate Q cu V
 pentru fiecare u∈V executa
      d[u] = ∞; tata[u]=0
  d[s] = 0
  cat timp Q ≠ ∅ executa
      u = extrage vârf cu eticheta d minimă din Q
      pentru fiecare uv∈E executa
           daca d[u]+w(u,v)<d[v] atunci
                  d[v] = d[u]+w(u,v)
                  tata[v] = u
  scrie d, tata

  //scrie drum minim de la s la t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

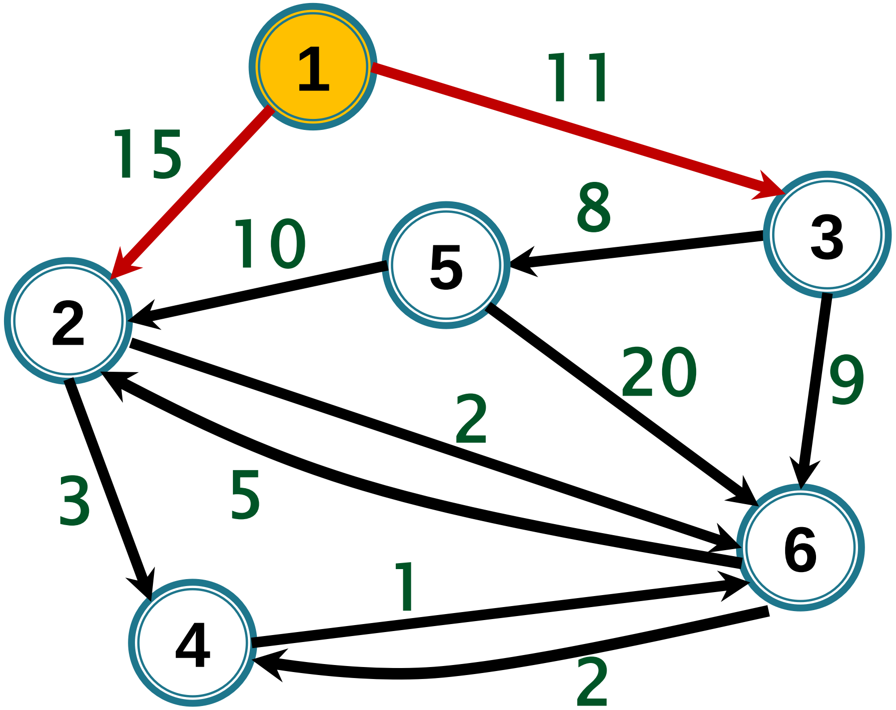
- **Observație**

  Vom demonstra că atunci când u este extras din Q eticheta lui d[u] este chiar cu δ(s,u) (este corectă) şi **nu se va mai actualiza** $\Rightarrow$ $\cancel{v \in Q}$

Exemplu

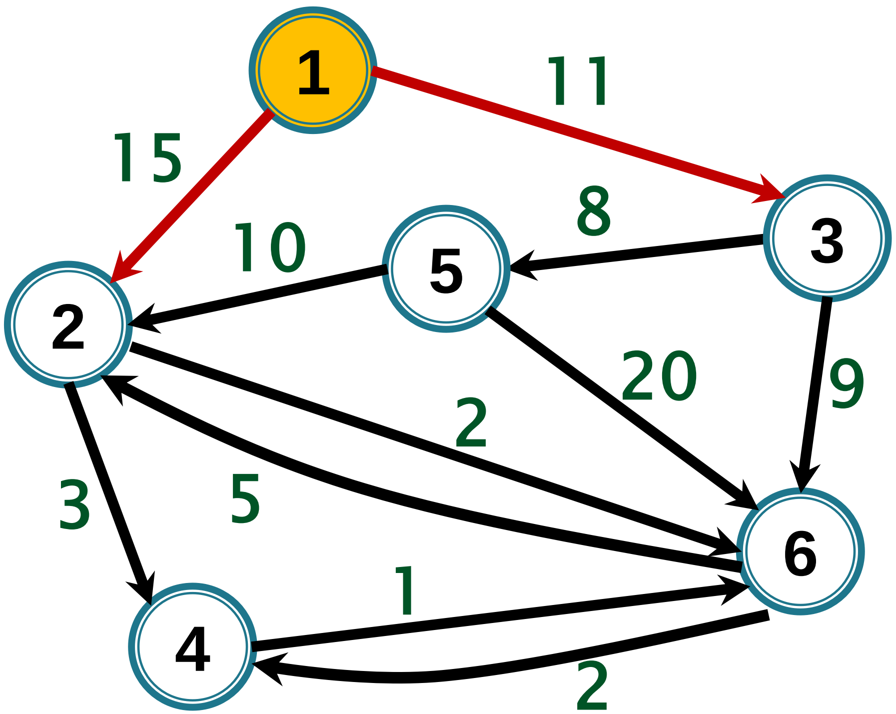


d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]



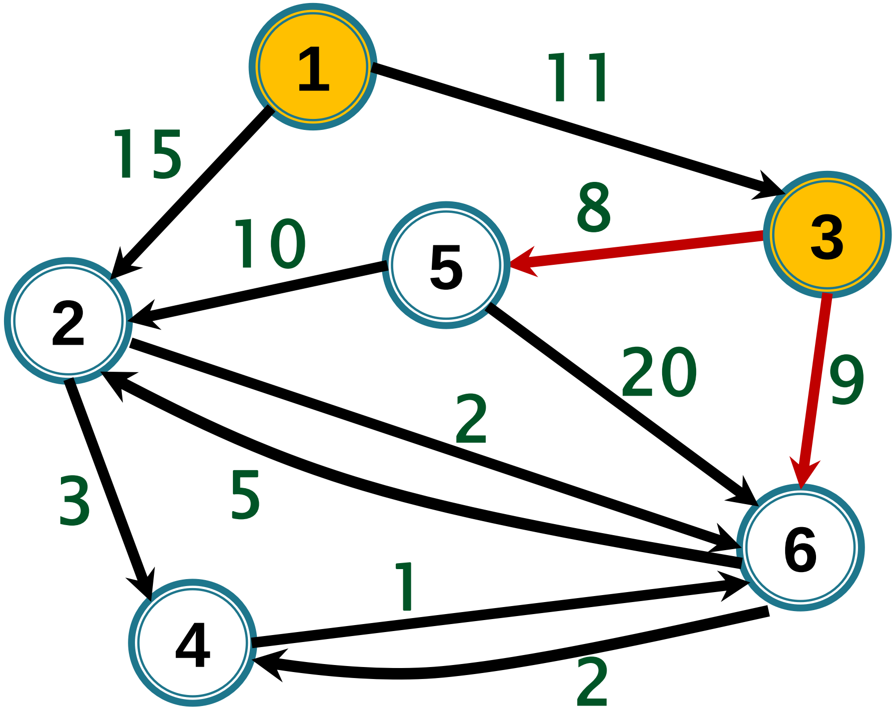
d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:						

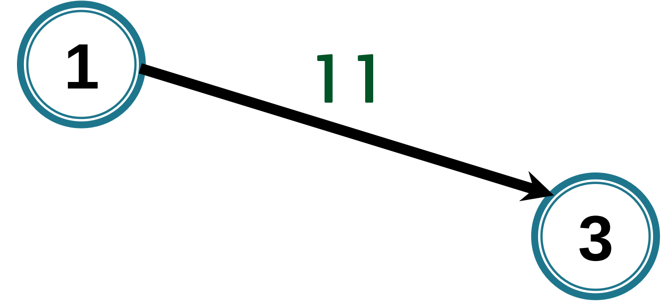
d[v] = min{d[v],d[u]+w(u,v)}



d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]

d[v] = min{d[v],d[u]+w(u,v)}

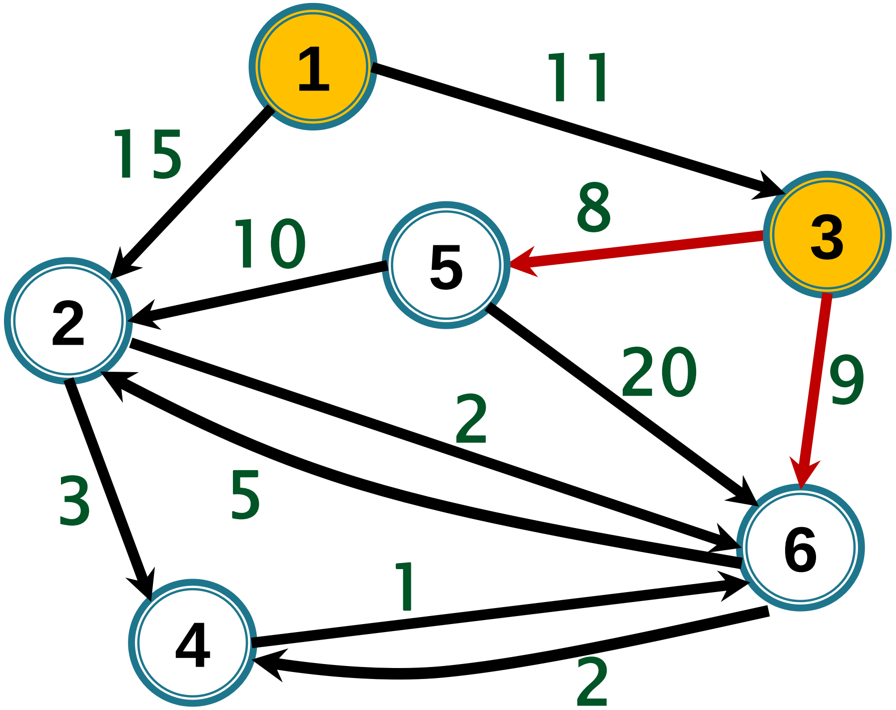


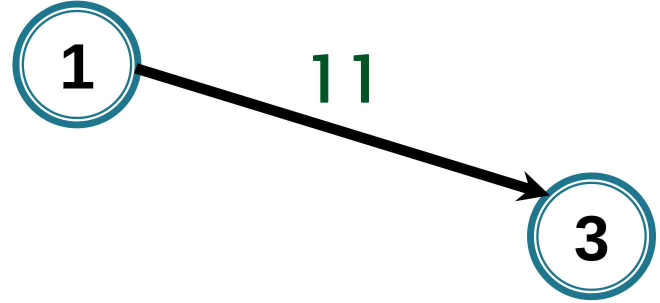


d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3						

d[5] = min{d[5],d[3]+w(3,5)}

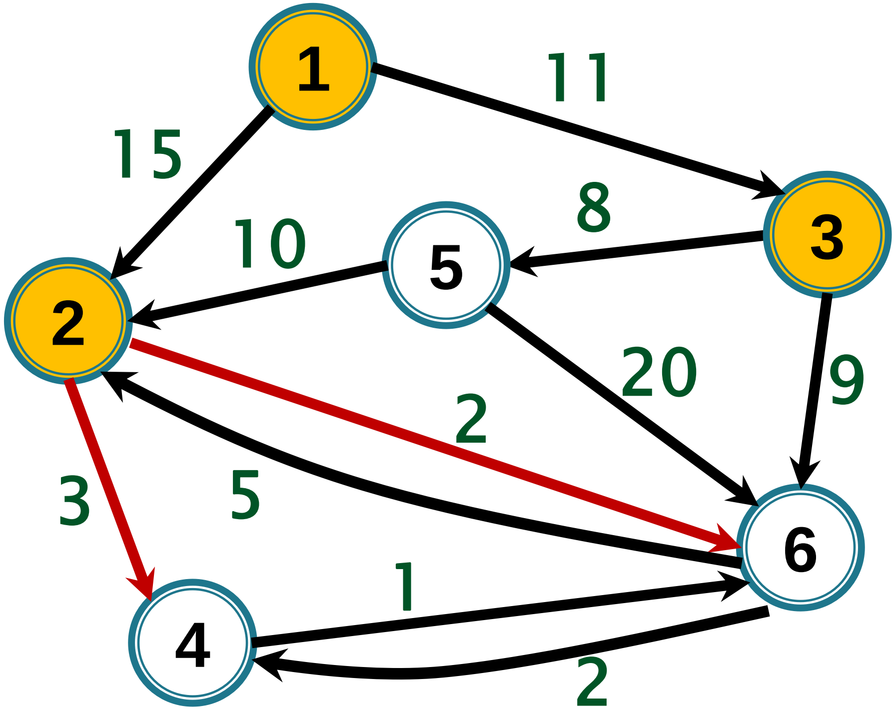
d[v] = min{d[v],d[u]+w(u,v)}

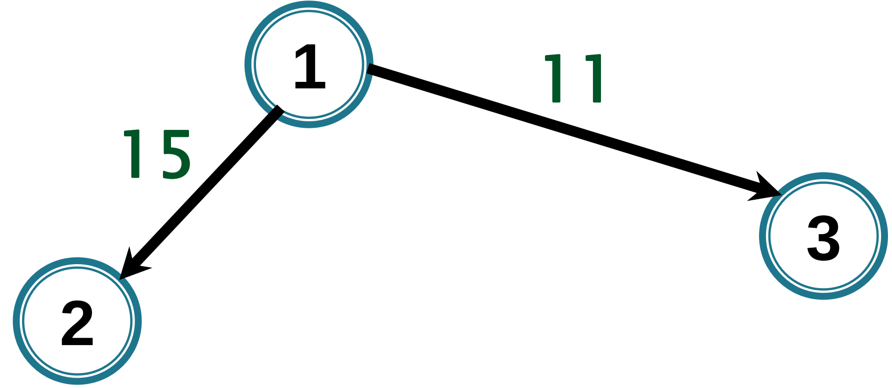




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]

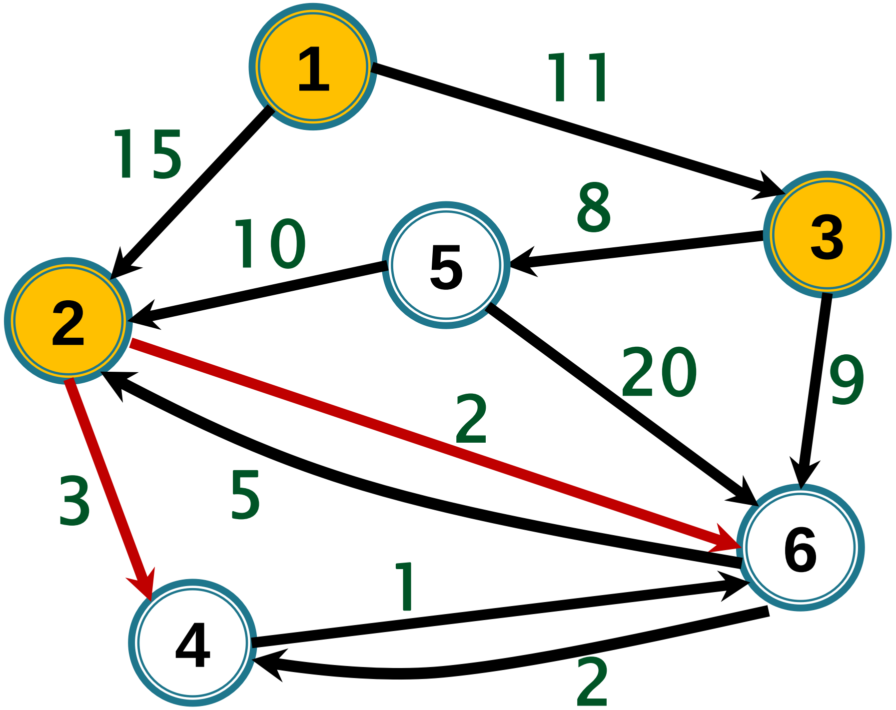
d[v] = min{d[v],d[u]+w(u,v)}

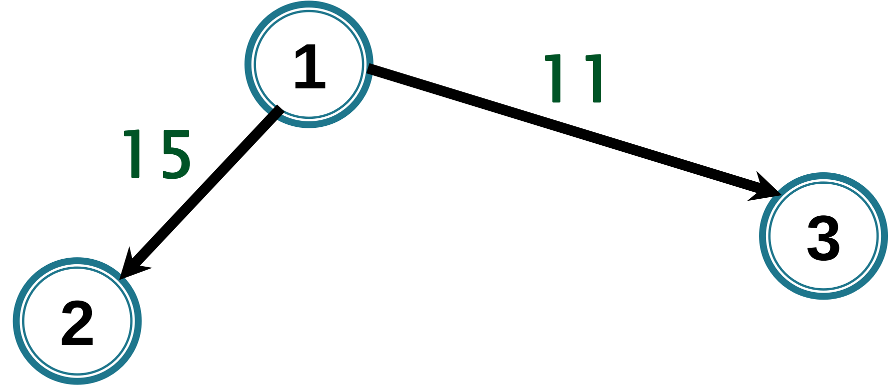




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:						

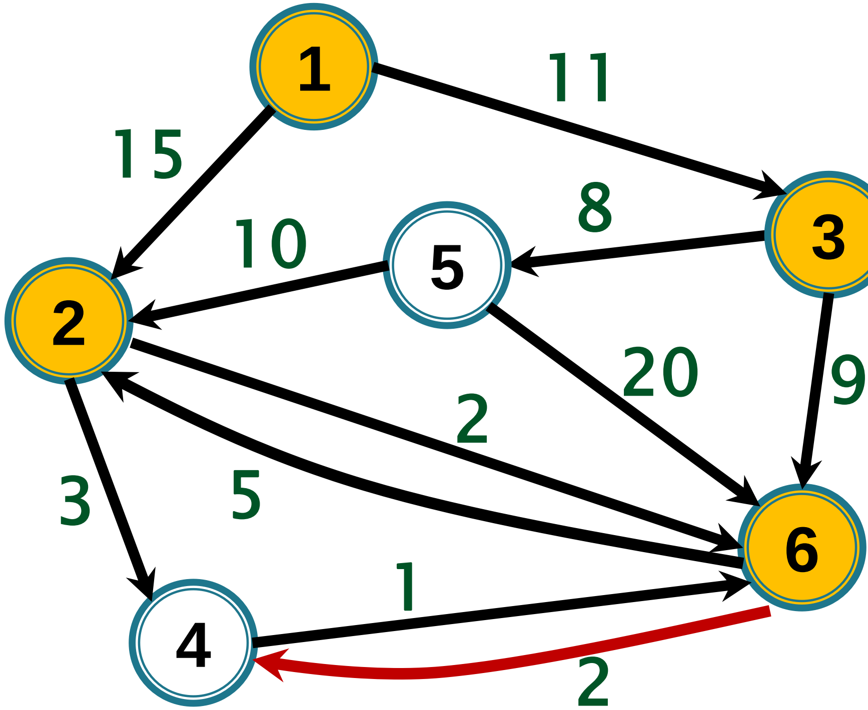
d[v] = min{d[v],d[u]+w(u,v)}

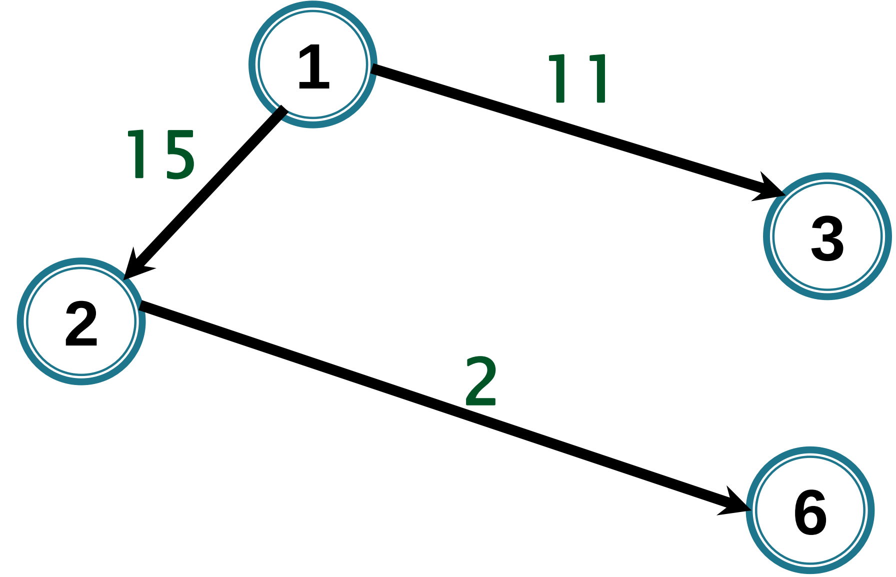




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]

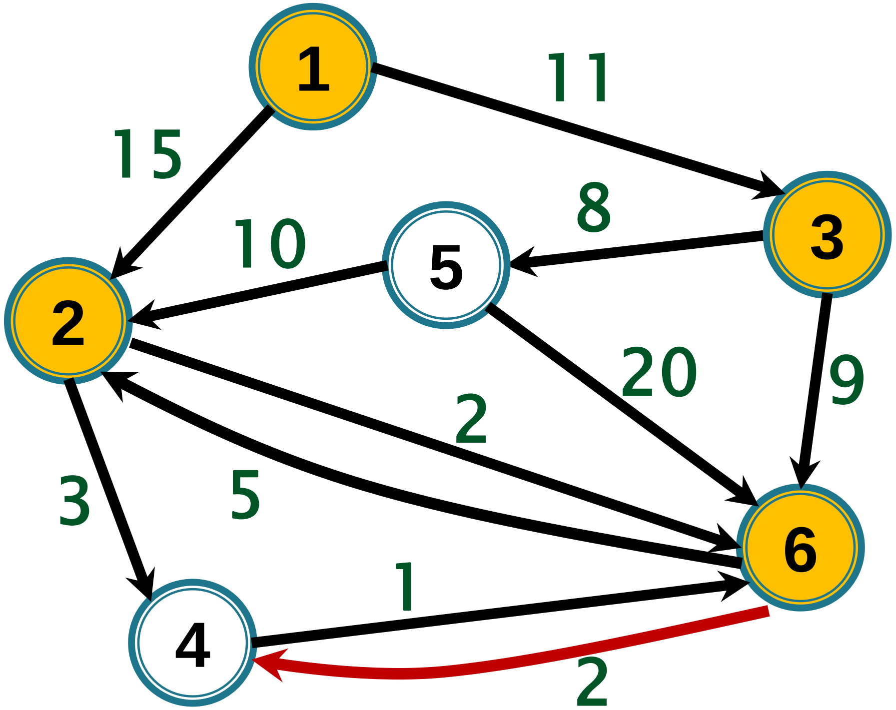
d[v] = min{d[v],d[u]+w(u,v)}

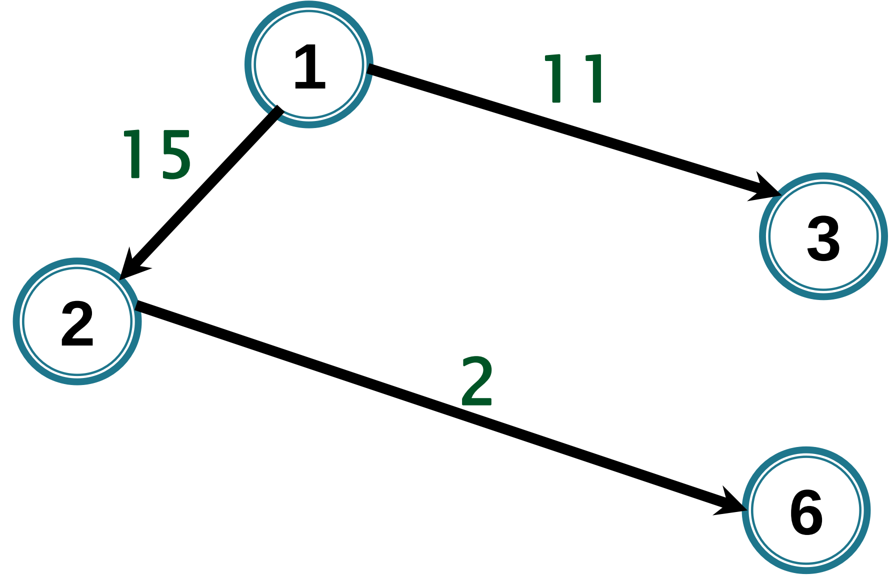




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:						

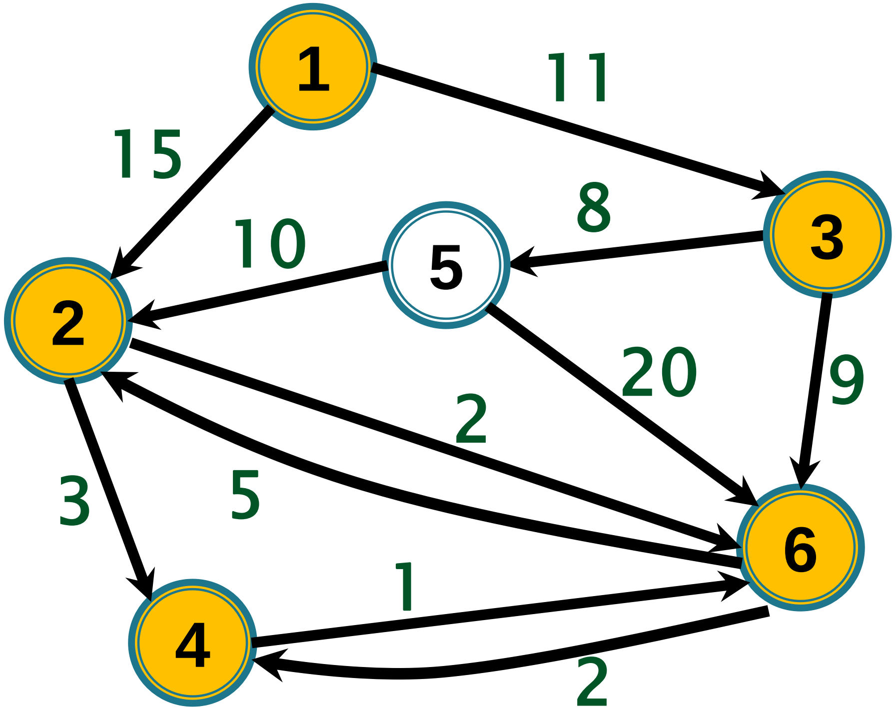
d[v] = min{d[v],d[u]+w(u,v)}

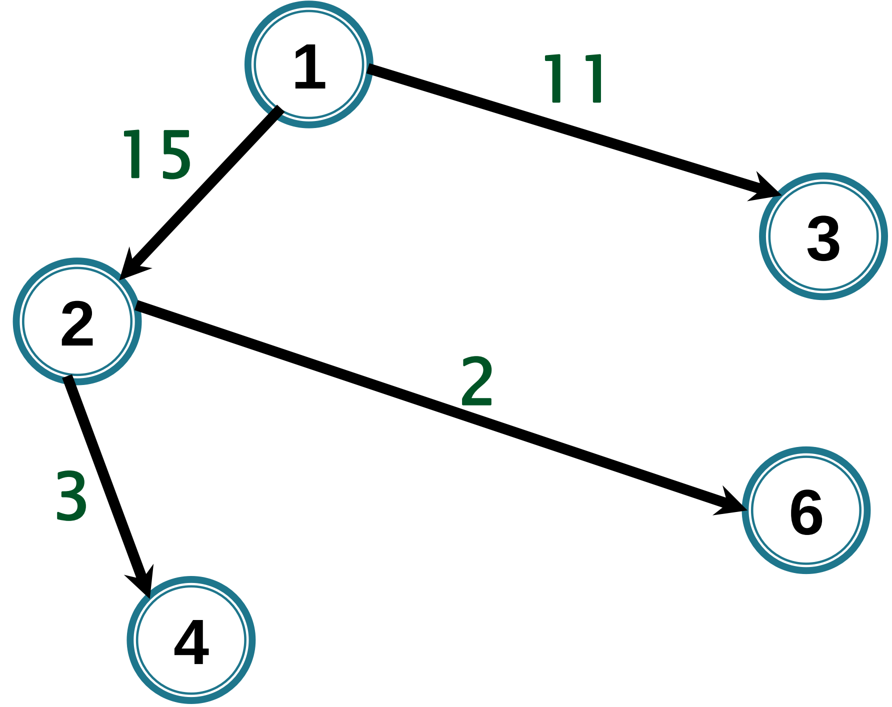




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]

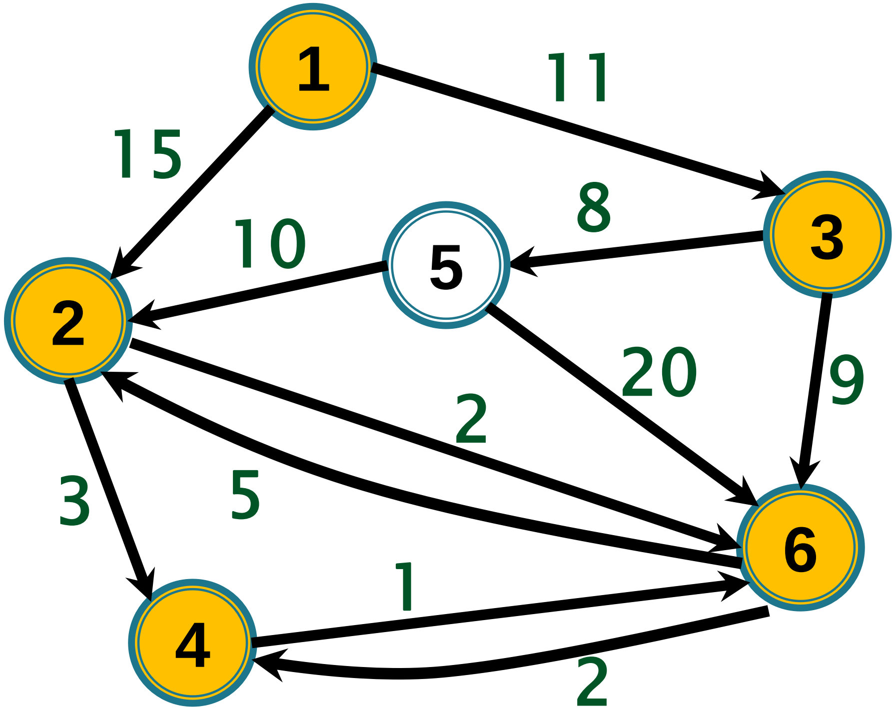
d[v] = min{d[v],d[u]+w(u,v)}

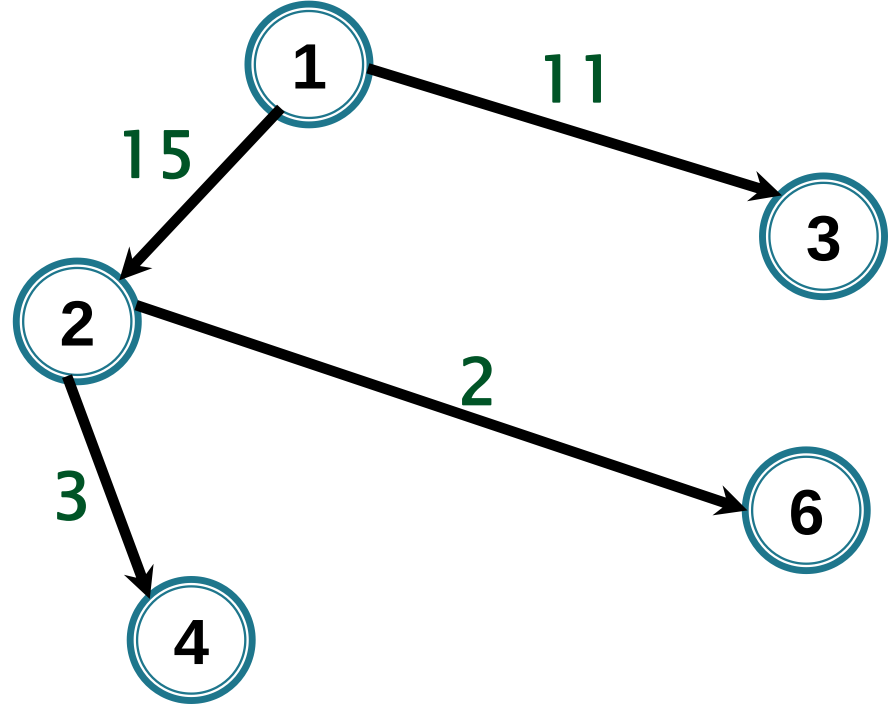




d/tata		1	2	3	4	5	6	
	[	0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0	]
Sel. 1:	[	– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0	]
Sel. 3:	[	– ,	15/1,	– ,	∞/0,	19/3,	20/3	]
Sel. 2:	[	– ,	– ,	– ,	18/2,	19/3,	17/2	]
Sel. 6:	[	– ,	– ,	– ,	18/2,	19/3,	–	]
Sel. 4:								

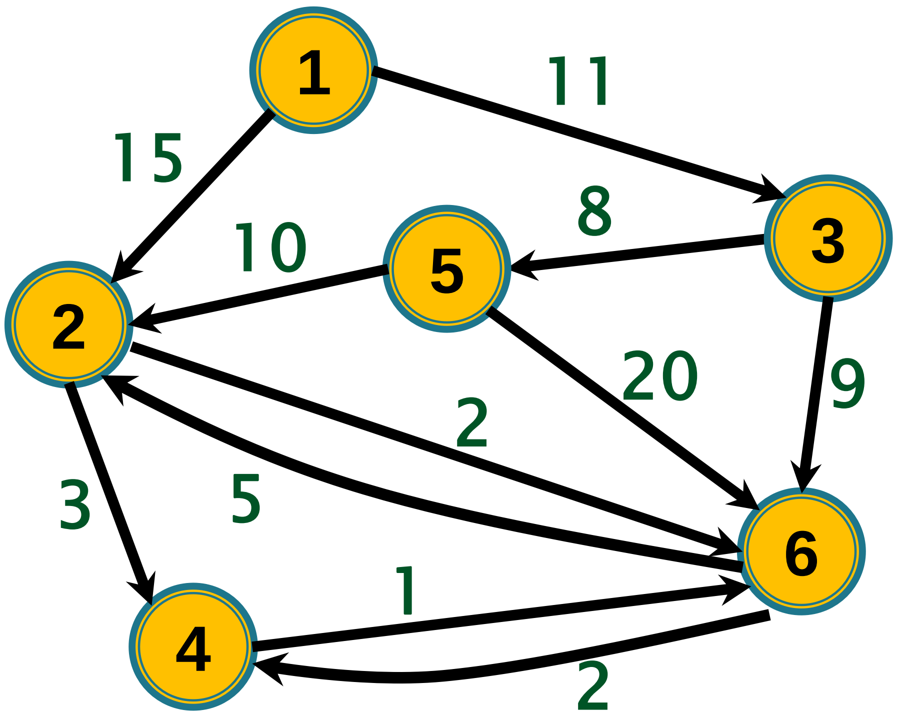
d[v] = min{d[v],d[u]+w(u,v)}

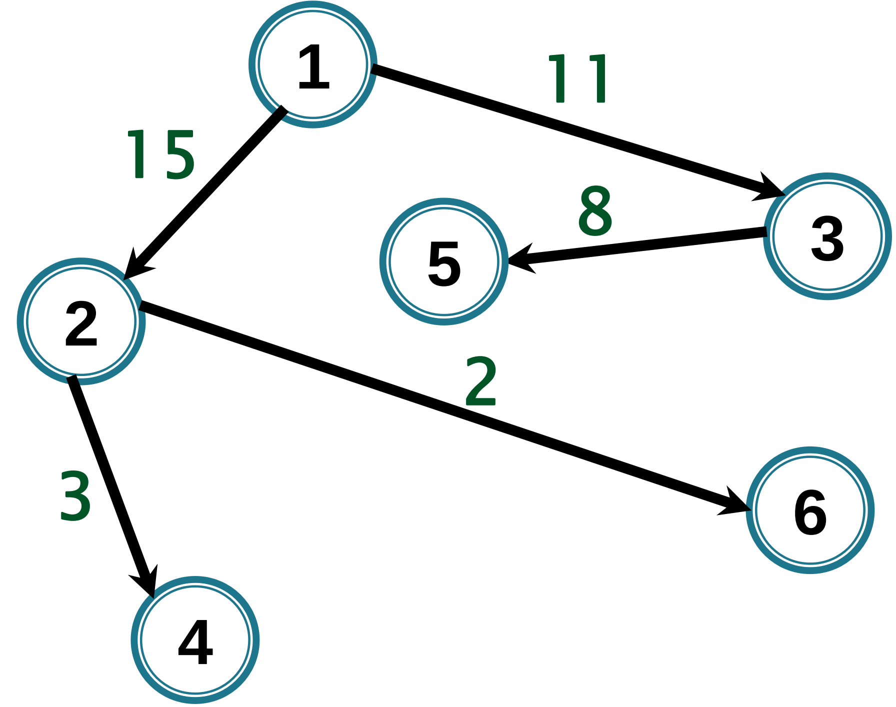




d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]

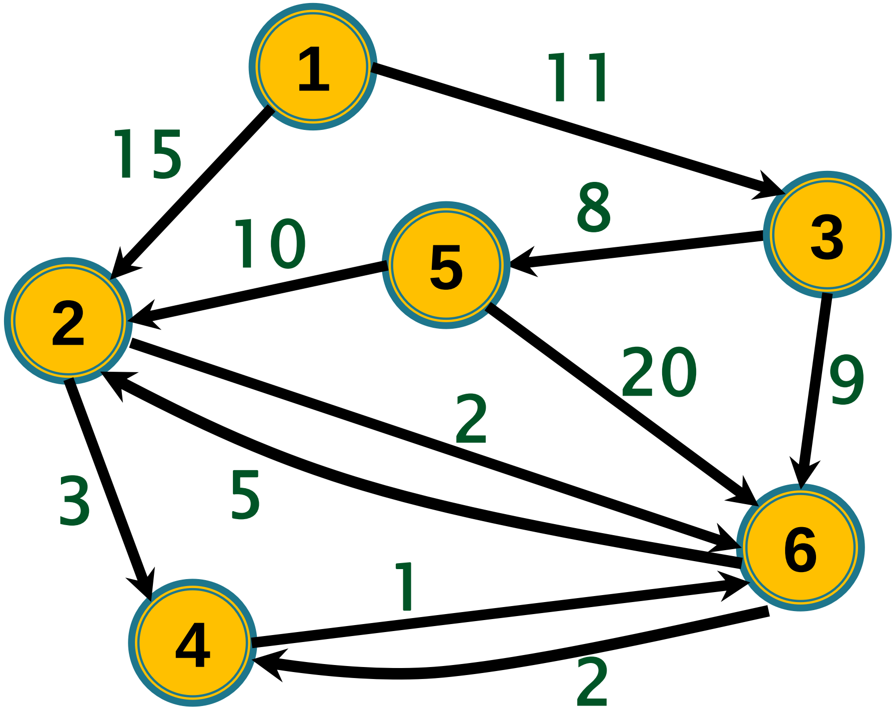
`d[v] = min{d[v],d[u]+w(u,v)}`

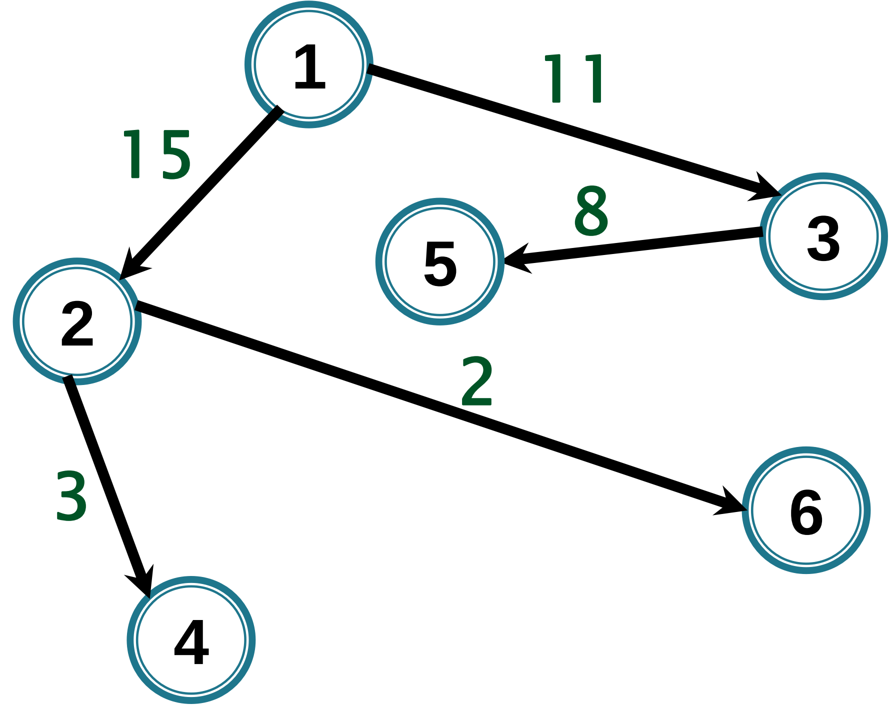




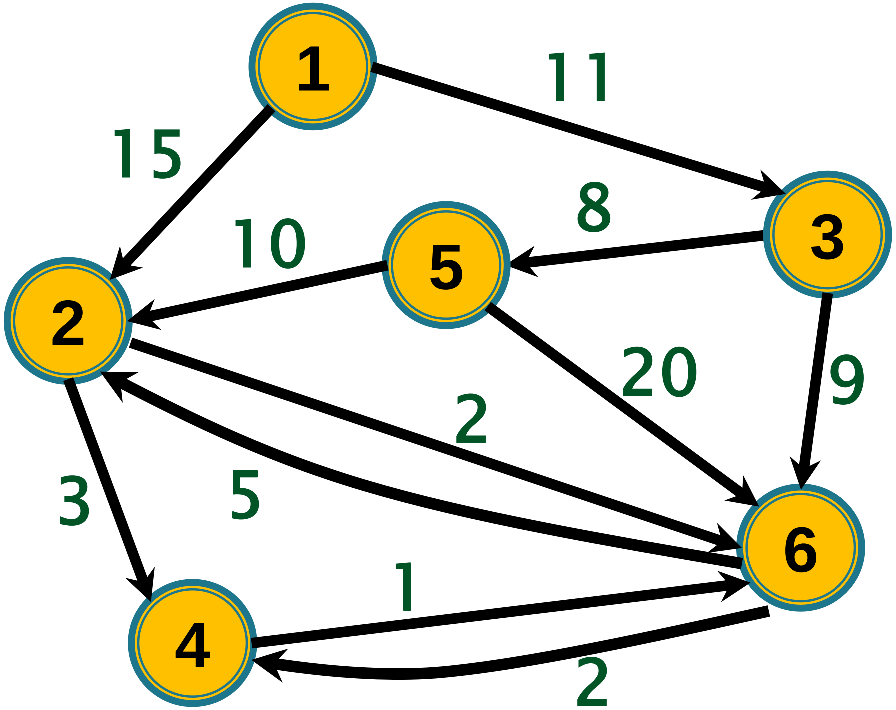
d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]
Sel. 5:						

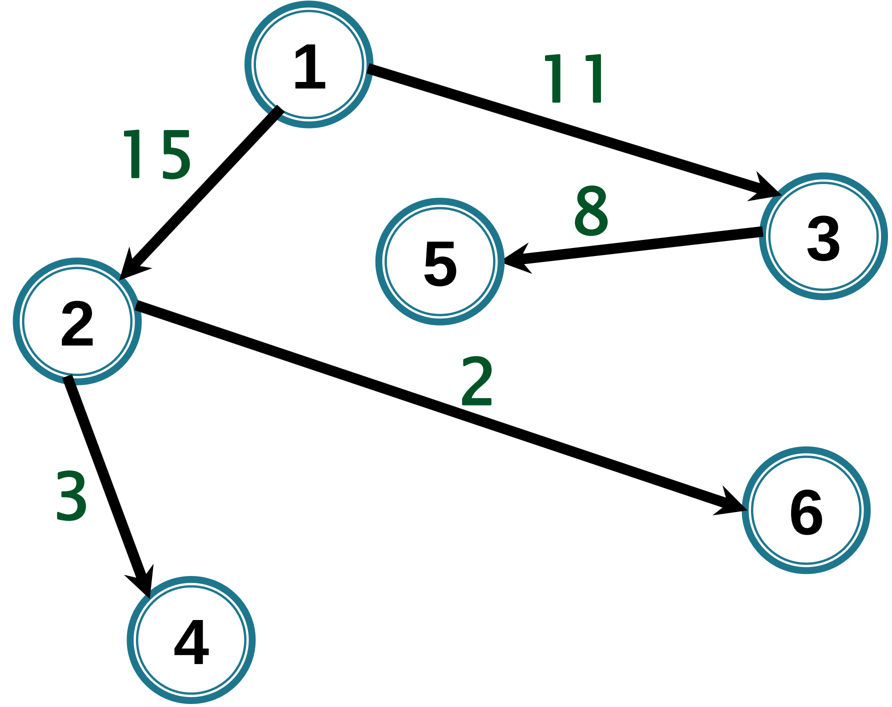
d[v] = min{d[v],d[u]+w(u,v)}





d/tata	1	2	3	4	5	6
	[0/0,	∞/0,	∞/0,	∞/0,	∞/0,	∞/0]
Sel. 1:	[– ,	15/1,	11/1,	∞/0,	∞/0,	∞/0]
Sel. 3:	[– ,	15/1,	– ,	∞/0,	19/3,	20/3]
Sel. 2:	[– ,	– ,	– ,	18/2,	19/3,	17/2]
Sel. 6:	[– ,	– ,	– ,	18/2,	19/3,	–]
Sel. 4:	[– ,	– ,	– ,	– ,	19/3,	–]
Sel. 5:	[– ,	– ,	– ,	– ,	– ,	–]





d/tata	1	2	3	4	5	6
Soluție:	[0/0,	15/1,	11/1,	18/2,	19/3,	17/2]

Un drum minim de la 1 la 6?

Algoritmul lui Dijkstra

- **Observații.**

1. Dacă vârful u curent are eticheta d[u] = ∞, algoritmul se poate opri

2. Vectorul tata memorează arborele distanțelor față de s
(vârfurile neaccesibile din s rămân cu tata 0)

Complexitate

Dijkstra(G, w, s)

```
initializează mulțimea vârfurilor neselectate Q cu V
 pentru fiecare u∈V executa
     d[u] = ∞; tata[u]=0
 d[s] = 0
 cat timp Q ≠ ∅ executa
     u = extrage vârf cu eticheta d minimă din Q
     pentru fiecare uv∈E executa
          daca d[u]+w(u,v)<d[v] atunci
               d[v] = d[u]+w(u,v)
               tata[v] = u
 scrie d, tata

 //scrie drum minim de la s la t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

Cum memorăm Q = vârfurile încă neselectate?

Algoritmul lui Dijkstra

Q poate fi (ca și în cazul algoritmului lui Prim)

- **vector:**

  $$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat } (u \notin Q) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **min-ansamblu (heap)**

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** ->
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** -> $O(n)$
- **n * extragere vârf minim** -> $O(n^2)$
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** -> $O(n)$
- **n * extragere vârf minim** -> $O(n^2)$
- **actualizare etichete vecini** -> $O(m)$

Algoritmul lui Dijkstra

Complexitate – reprezentarea lui Q ca vector

$$Q[u] = \begin{cases} 1, & \text{dacă } u \text{ este selectat în } V(T) \\ 0, & \text{altfel } (u \in Q) \end{cases}$$

- **Inițializare Q** -> $O(n)$
- **n * extragere vârf minim** -> $O(n^2)$
- **actualizare etichete vecini** -> $O(m)$

$O(n^2)$

Algoritmul lui Dijkstra

Complexitate – Q min-heap

- **Inițializare Q** ->
- **n * extragere vârf minim** ->
- **actualizare etichete vecini** ->

Dijkstra(G, w, s) - Q min-heap in raport cu d

```
pentru fiecare u∈V executa
    d[u] = ∞; tata[u]=0
d[s] = 0
Q = V //creare heap cu cheile din d
cat timp Q ≠ ∅ executa
    u = extrage_min(Q)
    pentru fiecare uv∈E executa
        daca d[u]+w(u,v)<d[v] atunci
              d[v] = d[u]+w(u,v)
              repara(Q,v)
              tata[v] = u
scrie d, tata

//scrie drum minim de la s la t un varf t dat folosind tata
```

Algoritmul lui Dijkstra

Complexitate – Q min-heap

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** -> O(n log n)
- **actualizare etichete vecini** ->

Algoritmul lui Dijkstra

Complexitate – Q min-heap

- **Inițializare Q** -> O(n)
- **n * extragere vârf minim** -> O(n log n)
- **actualizare etichete vecini** -> O(m log n)

  !!+ actualizare Q

O(m log n)

Algoritmul lui Dijkstra

- **Observație. Pentru a determina drumul minim între două vârfuri s și t date putem folosi algoritmul lui Dijkstra cu următoarea modificare:**
    - dacă vârful u ales este chiar t, algoritmul se oprește;
    - drumul de la s la t se afișează folosind vectorul tata (vezi BF)

Algoritmul lui Dijkstra

- **Dijkstra ≈ Prim** (versiunea $O(n^2)$/$O(m \log n)$)

Algoritmul lui Dijkstra

Algoritmul funcţionează şi pentru grafuri neorientate?

Algoritmul lui Dijkstra

- **De ce nu funcţionează corect algoritmul dacă avem arce cu cost negativ + exemplu?**

Algoritmul lui Dijkstra

- Cum putem rezolva problema dacă avem și arce de cost negativ?

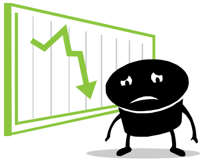
Algoritmul lui Dijkstra

- **Cum putem rezolva problema dacă avem și arce de cost negativ?**

Putem aduna o constantă la costul fiecărui arc astfel încât toate arcele să aibă cost pozitiv. Drumul minim între 2 vârfuri rămâne la fel?

Algoritmul lui Dijkstra

- Cum putem rezolva problema dacă avem și arce de cost negativ?
  - Putem aduna o constantă la costul fiecărui arc astfel încât toate arcele să aibă cost pozitiv. Drumul minim între 2 vârfuri rămâne la fel? – NU



Algoritmul lui Dijkstra

- Cum putem rezolva problema dacă avem și arce de cost negativ?

Algoritmul BELLMAN – FORD

Corectitudinea Algoritmului lui Dijkstra

Corectitudine

- **Lema 1.** Pentru orice $u \in V$, la orice pas al algoritmului lui Dijkstra avem:

  a) dacă $d[u] < \infty$, există un drum de la s la u în G de cost $d[u]$ și acesta se poate determina din vectorul tata:

  tata[u]= predecesorul lui u pe un drum de la s la u de cost d[u]

  b) $d[u] \geq \delta(s,u)$

Corectitudine

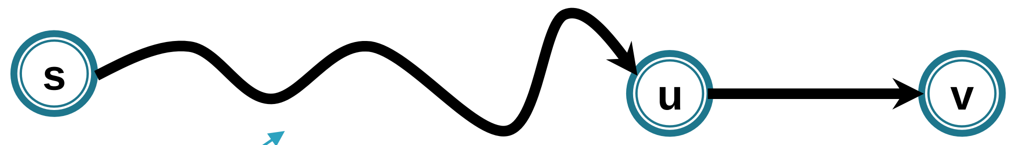
- **Demonstrație:** inducție după numărul de iterații (executii cat timp)

- Inițial d[s] = 0 = w([s]) (restul etichetelor sunt ∞)

- La prima iterație este extras din Q vârful s – pentru el afirmația se verifică

Corectitudine

- **Demonstrație:** Idee inducție

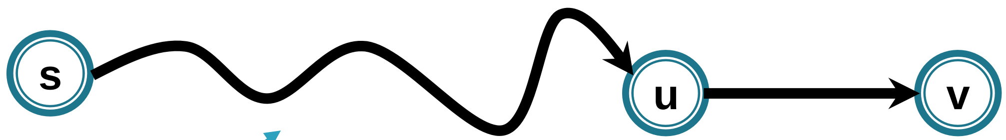


există P de la s la u cu w(P) = d[u]

relaxare uv

Corectitudine

- **Demonstrație:** Idee inducție



există P de la s la u cu w(P) = d[u]

După relaxare uv

$d[v] = d[u] + w(uv) = w(\,[s \underline{P}\, u;\, v])$

$tata[v] = u$

⇓

se verifică proprietatea și pentru v

Corectitudine

- **Lema 1.** Pentru orice $u \in V$, la orice pas al algoritmului lui Dijkstra avem:

  a) dacă $d[u] < \infty$, există un drum de la s la u în G de cost d[u] și acesta se poate determina din vectorul tata:

  tata[u]= predecesorul lui u pe un drum de la s la u de cost d[u]

  b) $d[u] \geq \delta(s,u)$

- **Consecinţă.** Dacă la un pas al algoritmului avem pentru un vârf u relația $d[u] = \delta(s, u)$, atunci d[u] nu se mai modifică până la final.

Corectitudine

- **Teoremă**

  Fie $G=(V, E, w)$ un graf orientat ponderat cu

  $w : E \rightarrow \mathbb{R}_+$ și $s \in V$ fixat.

  La finalul algoritmul lui Dijkstra avem:

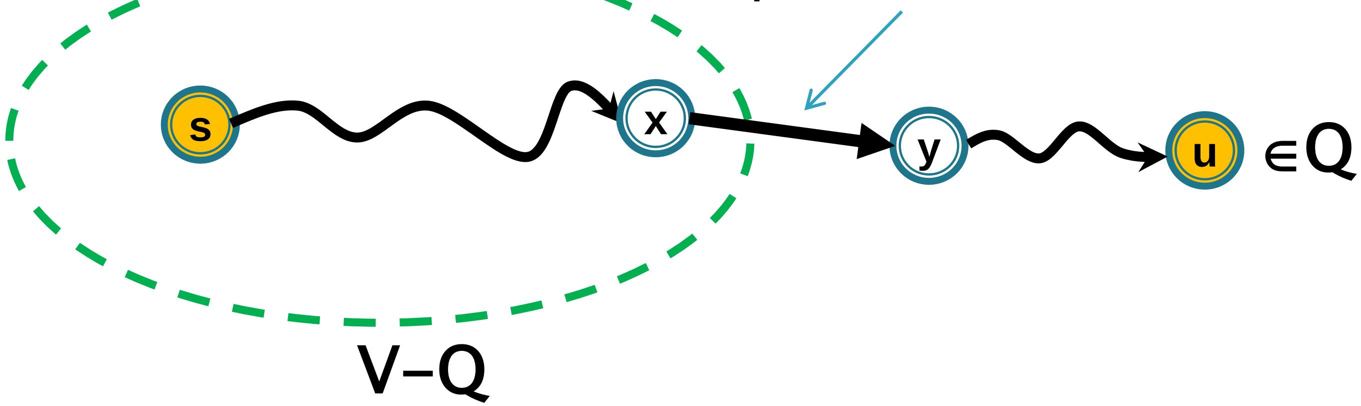
  $d[u] = \delta(s, u)$ pentru orice $u \in V$

  și tata memorează un arbore al distanțelor față de s.

Corectitudine

- **Demonstraţie (idee).** Inducţie: $d[x] = \delta(s, x)$ $\forall$ $x \notin Q$ (=deja selectat)
- Când un vârf u este selectat: fie P un s-u drum minim

primul arc de la V-Q la Q din P



dupa relaxarea lui xy (mai mult, are loc chiar egalitate: $d[y] = \delta(s,x) + w(x,y) = w\left(s \xrightarrow{P} y\right) = \delta(s,y)$)

din modul în care este ales u

$$d[u] \le d[y] \le d[x] + w(x,y) = \delta(s,x) + w(x,y) = w(s \xrightarrow{P} y)$$

$$\le w(P) \le d[u]$$

ipoteza de inductie pentru x

$$\Rightarrow d[u] = d[y] = w(P) = \delta(s,u)$$